06/12/98
PTO

WORKMAN NYDEGGER & SEELEY
ATTORNEYS AT LAW
A PROFESSIONAL CORPORATION
1000 EAGLE GATE TOWER
60 EAST SOUTH TEMPLE
SALT LAKE CITY, UTAH 84111
TELEPHONE (801) 533-9800
FACSIMILE (801) 328-1707

RICK D NYDEGGER
DAVID O SEELEY
BRENT P LORIMER
THOMAS R VUKSINICK
LARRY R LAYCOCK
JONATHAN W RICHARDS
DAVID R WRIGHT
JOHN C STRINGHAM
MICHAEL F. KRIEGER
BRADLEY K DESANDRO
JOHN M GUYNN
GREGORY M TAYLOR
DANA L TANGREN
KEVIN B. LAURENCE
ERIC L MASCHOFF
JEFFREY L RANCK

CHARLES J VEVERKA
ROBYN L PHILLIPS
DAVID B DELLENBACH
TIMOTHY M FARRELL
JOHN N GREAVES
VANESSA B. PIERCE
LENA I VINITSKAYA
KEVIN K JOHANSON
L DAVID GRIFFIN
R BURNS ISRAELSEN
MICHAEL T SANDERSON

H ROSS WORKMAN
OF COUNSEL

PATENT
TRADEMARK
COPYRIGHT
TRADE SECRETS
UNFAIR COMPETITION
LICENSING
COMPLEX LITIGATION

INTERNET
HOME PAGE: http://www wnspat com
GENERAL E-MAIL. Info@wnspat.com

A

Express Mail Certificate No. EL141420124US

PATENT APPLICATION
Docket: 13768.73

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Box: PATENT APPLICATION
Assistant Commissioner for Patents
Washington, DC 20231

Sir:

Transmitted herewith for filing is the patent application of:

Inventor:	Richard C. Machin
Citizenship:	United Kingdom
Address:	3530 169th Ave. NE Bellevue, WA 98008
Inventor:	Jameel Hyder
Citizenship:	India
Address:	23292 NE 16th Place Redmond, WA 98053
For:	METHOD, SYSTEM, AND COMPUTER PROGRAM PRODUCT FOR REPRESENTING AND CONNECTING AN UNDERLYING CONNECTION-ORIENTED DEVICE IN A KNOWN FORMAT

Comprising 54 pages of specification and claims.

Enclosed also are:

X Nineteen (19) sheets of formal drawings.

___ Declaration, Power of Attorney, and Petition.

___ Assignment of the invention.

___ A Verified Statement to Establish Small Entity Status Under 37 C.F.R. § 1.9 and 37 C.F.R. § 1.27.

_X_ A Certificate of Express Mailing by "Express Mail" certifying a filing date of June 12, 1998 by use of Express Mail Label No. EL141420124US.

___ Check for filing fees.

Please address all future correspondence in connection with the above-identified patent application to the attention of the following:

Rick D. Nydegger
WORKMAN, NYDEGGER & SEELEY
60 East South Temple
1000 Eagle Gate Tower
Salt Lake City, Utah 84111
Telephone: (801) 533-9800
Facsimile: (801) 328-1707

DATED this 12th day of June, 1998.

Respectfully submitted,

Timothy M. Farrell

TIMOTHY M. FARRELL
Attorney for Applicant
Registration No. 37,321

WORKMAN, NYDEGGER & SEELEY
1000 Eagle Gate Tower
60 East South Temple
Salt Lake City, Utah 84111
Telephone: (801) 533-9800
Fax: (801) 328-1707

TMF:cia
G:\DATA\WPDOCS2\CIA\TMF\MS\1376873.TRX

Express Mail Label No.: EL141420124US

PATENT APPLICATION
Docket: 13768.73

CERTIFICATE OF MAILING BY "EXPRESS MAIL"

"Express Mail" Mailing Label No.: EL141420124US

Date of Deposit: June 12th, 1998

I hereby certify that the following documents are being deposited with the United States Postal Service "Express Mail Post Office to Addressee" service under 37 C.F.R. § 1.10 on the date indicated above in an envelope addressed to Box PATENT APPLICATION, Commissioner of Patents and Trademarks, Washington, D.C. 20231:

- Patent application in the name of Richard C. Machin and Jameel Hyder, for Method, System, and Computer Program Product for Representing and Connecting an Underlying Connection-Oriented Device in a Known Format (54 pgs.)
- Formal Drawings (Nineteen (19) sheets)
- Transmittal Letter (2 pgs.)
- Postcard

Dated this 12th day of June, 1998.

Respectfully submitted,

Timothy M. Farrell

TIMOTHY M. FARRELL
Attorney for Applicant
Registration No. 37,321

WORKMAN, NYDEGGER & SEELEY
1000 Eagle Gate Tower
60 East South Temple
Salt Lake City, Utah 84111
Telephone: (801) 533-9800

TMF:cia
G:\DATA\WPDOCS2\CIA\TMF\MS\1376873.COD

Express Mailing Label No. EL141420124US

PATENT APPLICATION
Docket No. 13768.073

UNITED STATES PATENT APPLICATION

of

Richard C. Machin

and

Jameel Hyder

for

Method, System, and Computer Program Product for Representing and Connecting An Underlying Connection-Oriented Device in a Known Format

WORKMAN, NYDEGGER & SEELEY
A PROFESSIONAL CORPORATION
ATTORNEYS AT LAW
1000 EAGLE GATE TOWER
60 EAST SOUTH TEMPLE
SALT LAKE CITY, UTAH 84111

BACKGROUND OF THE INVENTION

1. The Field of the Invention

The present invention relates to methods for managing connection-oriented hardware media in an I/O subsystem of a computer operating system. More particularly, the present invention relates to a proxy client of an integrating component that provides a generalized interface for connection management so that characteristics of the underlying connection-oriented devices may be represented by the proxy client to applications in a format known by the application program, such as Telephony Application Programming Interface (TAPI) line devices.

2. Present State of the Art

The effectiveness of general purpose stand alone computers, such as the personal computer found in most office environments and laptop computers increasingly used by professionals requiring portability, has been substantially improved by allowing communications between machines over a communications network. Such networking of computers allows the sharing of resources found on one computer with other computers in the network. For example, storage areas having files, printers, modems, and other resources may all be advantageously shared.

Data that is shared between computers can be sent in packets across a physical network connection and read by destination computers. Such packetized network data may be requests for shared resources, data, such as a file, or other information that must be communicated from one computer to the other. As used herein, the term "network data" refers to data or information that is actually transmitted over the communications network between different computers.

The physical network between the different computers can be categorized into two general types. The first type of network makes use of a "connectionless" media wherein the

packetized network data contains destination information, such as a network address, and is simply placed on the network by the hardware media (hereinafter referred to simply as "media") and routed to the destination. A common network interface card that provides access to an Ethernet would be an example of a connectionless media. The destination will recognize the address in the packet and process it accordingly while other destination nodes will simply ignore the packet since it lacks the correct address information.

Another type of physical network utilizes a connection-oriented hardware media, such as telephone modem, cable modem, or ISDN connector. With connection-oriented media, a connection must be made and maintained in addition to sending and receiving packets of information. The connection-oriented media will send and receive data packets directly with the destination node that has matching hardware and with which a connection has been made. Once data is received, it is treated in the same manner by the I/O subsystem. Connection-oriented media are commonly found for Wide Area Network (WAN) implementations.

On a particular computer or node of the network, a network interface card (NIC) or network card monitors the physical communications channel for packets destined for that computer as well as transmits packets of network data destined for other computers. A connection-oriented NIC will first have to make the connection before packets may be sent and received. Once the connection is made, the connection-oriented NIC may receive and send packets as described above depending on the nature of the connection-oriented media. Software components run on the node computer under direction or control of the operating system or architecture for managing and controlling the network card operations. Furthermore, other software components exist to further abstract the network communications channel and provide more and more general networking interfaces for higher layers using their services. The layered approach allows compartmentalization and easier development of network applications.

WORKMAN, NYDEGGER & SEELEY
A PROFESSIONAL CORPORATION
ATTORNEYS AT LAW
1000 EAGLE GATE TOWER
60 EAST SOUTH TEMPLE
SALT LAKE CITY, UTAH 84111

One model used to provide a structure for layered software component development is the seven-layer ISO model that is well known in the art. While actual implementations of the ISO model do not necessarily rigidly isolate each particular layer as a separate component exposing its own interface to layers above and below, the concepts of the model are generally applicable. For purposes of this disclosure, the lower layers of the ISO model are most at issue, namely, the data link layer implemented by a network card device driver, and the transport and network layers implemented as a transport protocol driver.

Lower level networking functions, such as are discussed throughout this application with respect to controlling a network card and initial processing of packetized network data, are handled by special system software components called drivers that integrate with a host operating system according to a specific architecture and have special privileges for accessing system resources. Throughout this application, reference will be made to the Windows NT® operating system available from Microsoft Corporation and to its specific architecture wherein lies one embodiment of the present invention. Such drivers run in "kernel mode," meaning they have higher privileges and access to system resources than do "user mode" application process threads. While specific reference is made to Windows NT concepts and terminology, those skilled in the art will recognize that many, if not most, operating systems share similarities relevant to the environment of the present invention.

Because there are different types of transport protocols developed over time by different entities for different reasons, there may be different types of transport protocol drivers acting as software components running on a single host computer system in order to provide the necessary networking capabilities for a given installation. Some common transport protocols include TCP/IP, IPX, AppleTalk®, and others. Each transport protocol driver will communicate with one or more individual network card device drivers in order to send network data over a communications network and receive incoming packets from the communications network.

WORKMAN, NYDEGGER & SEELEY
A PROFESSIONAL CORPORATION
ATTORNEYS AT LAW
1000 EAGLE GATE TOWER
60 EAST SOUTH TEMPLE
SALT LAKE CITY, UTAH 84111

Furthermore, because there are a multitude of network cards provided by numerous manufacturers, there are a corresponding large number of potential network card device drivers. In order to support full connectivity to the transport protocol drivers, each network card device driver must support the ability to communicate with each different type of transport protocol driver. Because of the complexity of many different variations that could conceivably be connected together due to the layered component approach, building such drivers can be a time intensive process and the nature of the different interfaces each driver must use is illustrated in Figure 1.

Figure 1 is a block diagram showing the structure of a plurality of network cards, network card device drivers, and transport protocol drivers that each must interact with system resources and a central database or registry having connectivity information in order to operate properly. Furthermore, each transport protocol driver must support each and every network card device driver for which it may be connected and in like manner each network card device driver must support communicating with each and every transport protocol driver to which it may be connected.

If a new transport protocol driver is introduced, each network card device driver wanting to support the new transport protocol driver may require modification to the source code followed by a re-release and distribution of the executable driver code. Likewise, a new network card device driver may also require a similar re-release. Releasing and distributing software is an expensive process that software companies desire to limit as much as possible.

For example, passing network information arriving on network card 20 controlled by network card device driver 22 to the transport protocol driver 24 requires the transport protocol driver 24 and the network card device driver 22 to be fairly complex in terms of programming effort. This may take significant time for a developer or engineer to create. Note that the network card driver 22 must not only interact with the network interface card 20 but also have an interface 26 to the system resources 28 as well as an interface 30 to the

registry 32 containing connectivity information. Through such interfaces and the programming entailed therein, the network card device driver 22 will receive an interrupt that a packet has been received or is available for receipt by having the system execute code in an interrupt handling routine previously registered that makes use of system resources such as RAM for storing the packet.

Furthermore, the network card device driver 22 will use the registry interface 30 to access the registry 32 connectivity information for determining which transport protocol driver(s) will receive the packetized network information. For purposes of this example, the transport driver 24 is the recipient as illustrated by connecting line 34. Note also that the network card device driver 22 must support or be able to communicate with other transport protocol drivers since a variety exist and it is not known at development time which transport protocol driver will be indicated in the control information found in the registry 32 for receiving the network data.

On the other hand, the protocol transport driver 24 must also interface with the system resources 28 and the registry 32 containing connectivity information. Again, in order to support the many available network card device drivers, each transport protocol driver will be a relatively complex software component since the precise network card device driver for interfacing is not known at the time of development.

One advance in the art that has reduced the complexity associated with developing transport protocol drivers and network card device drivers is that of an integrating component that provides an abstracted interface to transport protocol drivers developers and to network card device driver developers. Figure 2 is a block diagram showing the introduction of an integrating component that reduces the complexity of transport protocol driver development and network card device driver development. In such an environment, an integrating component 36 will have a registry interface 38 for accessing a registry 32 of connectivity information and a system resource interface 40 for accessing system resources 28. Therefore,

development of the network card device driver 42 for controlling network card 20 is greatly simplified. The network card device driver 42 must only support an interface 44 to the integrating component 36. In like manner, the transport protocol driver 46 is also further simplified as only an interface 48 to the integrating component 36 may be supported.

The complexity of interfacing directly with the system resources 26 and the registry 32 of connectivity information is now handled by the integrating component 36. Furthermore, the integrating component provides an interface to developers incorporating many services and functionality that will be common to network card drivers and transport protocol drivers allowing the drivers to be developed more efficiently.

Another inherent benefit is that all routing of packets between transport protocol drivers and network card device drivers is managed by the integrating component. A particular transport protocol driver or network card device driver does not need to know the specific interface of the other components processing the same network packet. In other words, any network card device driver written to the integrating component 36 will be able to communicate with any available transport protocol that is also written to the integrating component 36 as determined by the connectivity information contained in the registry 32 and vice versa with respect to transport protocol drivers communicating with network card device drivers.

Besides providing quicker transport network card device driver development, the use of an integrating component 36 also facilitates multi-platform support. The integrating component interface may be supported on many different platforms, effectively encapsulating the details of actual interfacing with a particular operating system and environment. A driver developer generally needs to write the driver only one time and simply recompile the driver on any system that has the integrating component 36 supported thereon.

One technology for integrating network card device drivers to transport protocol drivers is the Network Driver Interface Specification (NDIS) technology implemented on the

WORKMAN, NYDEGGER & SEELEY
A PROFESSIONAL CORPORATION
ATTORNEYS AT LAW
1000 EAGLE GATE TOWER
60 EAST SOUTH TEMPLE
SALT LAKE CITY, UTAH 84111

Windows NT operating system as the NDIS wrapper device driver. The NDIS technology is also supported on other systems, such as the Windows95® operating system, in order to support cross platform support of network card device drivers and transport protocol drivers. The integrating component manages all interaction with system level services and hardware to further reduce development complexity of connected drivers. For example, the NDIS wrapper manages initial interrupt processing, system memory allocations to connected drivers, allocation to other hardware resources, etc. as well as providing packet routing capability between network card device drivers and transport protocol drivers.

Referring now to Figure 3, a logical diagram showing a number of different parts of software for a connection-oriented hardware media that utilizes an integrating component, such as integrating component 36 explained previously, is presented. Figure 3 represents one way of handling connection-oriented hardware media that utilizes an integrating component but where the connection-oriented device driver must still provide a connection interface and connection management functionality that must be replicated for each and every connection-oriented device driver. In other words, every connection-oriented hardware manufacturer must develop and provide, as part of the device driver, the connection management ability. Furthermore, in many instances an application must be programmed to a proprietary connection interface which further limits the flexibility having an integrating component.

Connection-oriented hardware adapter 52 provides access to a certain media type and is controlled by the connection-oriented device driver 54 as represented by arrow 56. This control includes all the connection creation in management control as well as the packetized network data control that is not associated with the integrating component 58. The connection-oriented device driver 54 communicates with the integrating component 58 as indicated by arrow 60 in the same manner as explained previously in Figure 2 to provide a data path and to a connection-oriented data transport 62 (as indicated by arrow 64).

WORKMAN, NYDEGGER & SEELEY
A PROFESSIONAL CORPORATION
ATTORNEYS AT LAW
1000 EAGLE GATE TOWER
60 EAST SOUTH TEMPLE
SALT LAKE CITY, UTAH 84111

Finally, the application 66 communicates with the connection-oriented data transport 62, again, as indicated by arrow 68. Note that the arrows used to indicate communication between the various components may in fact indicate communication through additional components. For example, communication may be through an operating system or there may be additional components. Figure 3 is simplified in order to focus on the two different channels that an application 66 would use to manage connection-oriented hardware.

Arrow 68 may consist of a path of various components in transporting data to and from the data transport 62 that are unimportant for this discussion. For example, the application may communicate with a WinSock communications component that may communicate with yet other components before data arrives at the data transport. All such components are incorporated as part of arrow 68.

Besides the data channel through the data transport 62 in the integrating component 58, the connection-oriented device driver 54 provides a connection channel to the application 66 by means of a connection interface 72 that allows communication between the application 66 and the connection-oriented device driver 54 as indicated by arrow 74. This connection interface 72 can be either proprietary or standardized but in either case must be provided by the connection-oriented device driver 54.

Furthermore, the actual connection management functionality 76 is also included in the connection-oriented device driver 54 thereby increasing significantly in some instances the amount of development for a connection-oriented device driver 54. The connection management functionality 76 includes media-specific control and protocol information for creating and managing a connection over the media by the connection-oriented hardware adapter 52. For example, for Asynchronous Transfer Mode (ATM) media, a certain set of signaling protocols and control information is used, regardless of the actual hardware created by different manufacturers. In other words, each manufacturer must create the same connection management protocol functionality for an ATM card as every other manufacturer.

WORKMAN, NYDEGGER & SEELEY
A PROFESSIONAL CORPORATION
ATTORNEYS AT LAW
1000 EAGLE GATE TOWER
60 EAST SOUTH TEMPLE
SALT LAKE CITY, UTAH 84111

This represents a redundant development effort, particularly when the connection management functionality constitutes a very large portion of the development of the connection-oriented device driver 54 and such connection management functionality is readily defined by adopted standards.

Referring now to Figure 4, a logical diagram is shown that illustrates the environment in the Microsoft NT operating system. This diagram is equivalent in its functional nature to that shown in Figure 3 but introduces concepts specific to the NT environment. Note that this is only one example of components used to make and use a connection in a connection-oriented architecture and others do exist.

The application program 78 will be written to communicate with a Win32 communications applications programming interface (API) 80, such as WinSock, as indicated by arrow 82 in order to send and receive data over a connection-oriented data channel with the appropriate connection-oriented transport as has been explained previously. For creating and maintaining a connection, the application 78 may communicate through a Telephony API 84 or TAPI 84 as indicated by arrow 86, or some other interface. Both the telephony API 84 and the Win32 communications API 80 reside in user mode for the NT operating system. Note that elements of the Win32 communications API 80 may be used for creating a connection.

A demarcation between user mode and kernel mode is indicated by dash line 88. Furthermore, there may be additional kernel mode components and/or user mode components that are used to make the communication channels as shown.

Connection-oriented hardware device driver 90 controls the connection-oriented hardware adapter 92 as indicated by arrow 94. For purposes of the data channel, the connection-oriented device driver 90 will communicate bi-directionally with the integrating component 96 as indicated by arrow 98. The data channel is completed by having the integrating component 96 communicate with the connection-oriented data transport 100 as

WORKMAN, NYDEGGER & SEELEY
A PROFESSIONAL CORPORATION
ATTORNEYS AT LAW
1000 EAGLE GATE TOWER
60 EAST SOUTH TEMPLE
SALT LAKE CITY, UTAH 84111

indicated by arrow 102 and further having the data transport 100 communicate with the Win32 API library 80 as indicated by arrow 104. The data channel aspect is relatively known in the art with respect to connectionless device drivers. Because the integrating component 96 incorporates much common functionality common to all network data communications, connectionless device drivers are written in a much simplified fashion.

Again, the connection channel may require substantial effort on the part of the device driver developer. For example, the connection-oriented device driver 90 must provide a connection interface 106 that will communicate with the telephoning API 84 as indicated by arrow 108. Every connection-oriented device driver developer will need to provide such an interface. Furthermore, the connection management functionality 110 that is specific to a particular media upon which the connection-oriented hardware communicates (*e.g.*, ATM, ISDN, etc.) can amount to a substantial amount of development effort. Again, such development effort is duplicate across many developers that essentially implement the same functionality.

Because of the added driver development effort required in order to provide a connection management interface that is either proprietary or that interfaces with existing application level interfaces (*e.g.*, TAPI), it would be desirable to provide such connection interface in an abstracted form so that the connection-oriented device driver development may be simplified. Furthermore, since applications are generally written to a higher level interface, such as TAPI, rather than to a lower-level I/O subsystem integrating component, such as NDIS, it would be highly beneficial to represent the connection functionality provided by the I/O subsystem in a way that is readily implemented by the higher level components, such as TAPI. This allows the device characteristics to be more immediately available to the applications without having the application be programmed to the driver subsystem interface. This is beneficial in that applications can take advantage of the extended connection capabilities with little or no modification.

WORKMAN, NYDEGGER & SEELEY
A PROFESSIONAL CORPORATION
ATTORNEYS AT LAW
1000 EAGLE GATE TOWER
60 EAST SOUTH TEMPLE
SALT LAKE CITY, UTAH 84111

WORKMAN, NYDEGGER & SEELEY
A PROFESSIONAL CORPORATION
ATTORNEYS AT LAW
1000 EAGLE GATE TOWER
60 EAST SOUTH TEMPLE
SALT LAKE CITY, UTAH 84111

SUMMARY AND OBJECTS OF THE INVENTION

It is an object of the present invention to take advantage of a connection-oriented I/O subsystem without requiring an application programmer to learn a new interface.

It is an object of the present invention to redirect data and data control information arranged in a certain format, such as TCP/IP, to be routed to an appropriate data transport while the connection control information continues to be routed through the original connection.

Additional objects and advantages of the invention will be set forth in the description which follows, and in part will be obvious from the description, or may be learned by the practice of the invention. The objects and advantages of the invention may be realized and obtained by means of the instruments and combinations particularly pointed out in the appended claims.

To achieve the foregoing objects, and in accordance with the invention as embodied and broadly described herein a method, and computer program product for representing and connecting an underlying connection-oriented device in a known format is provided. Note that co-pending U.S. Patent Application Serial No. _________ (Filed June 12, 1998) entitled "Method, Computer Program Product, and System for Separating Connection Management Functionality from a Connection-Oriented Device Driver," and U.S. Patent Application Serial No. _________ (Filed June 12, 1998) entitled "Method and Computer Program Product for Managing Connection-Oriented Media" are herein incorporated by reference.

The present invention allows an application to easily ascertain characteristics of connection-oriented media and create a connection without requiring the application programmer to use a low-level API, such as NDIS. Furthermore, the underlying device characteristics and connection may be created and managed through a known interface, such as TAPI. A code component, called a "proxy client component" or "proxy client", exposes underlying connection-oriented media as TAPI devices and receives TAPI requests to make

and control connections. The proxy client component is written to the integrating component interface and is able to create and maintain connections as well as know of existing data handlers registered to the integrating component. It does this by interacting with the integrating component as a data transport, connection manager component, and a client of the connection interface of the integrating component.

The proxy client component will handle translating a TAPI request into whatever interaction is necessary over an integrating component interface, such as the NDIS upper edge connection manager interface, to make and maintain a connection with the particular connection-oriented media. This frees the application programmer from any more programming complexity than knowledge of TAPI.

One TAPI request will ask that the data be passed to a certain type of handler or transport, such as TCP/IP, PPP, raw unframed data handler, etc. Such a handler or transport will have already have been written to the integrating component transport interface specification and will be "registered" with the integrating component making it "known" to the integrating component so that it may be exposed and made available over the connection manager interface to the proxy client component.

In response to the TAPI request that data be sent to a particular transport, the proxy client component will identify the correct transport, and interact with the integrating component in order to redirect data and data control information to the transport. The connection control information, however, will continue to flow through the proxy client component to the application as TAPI status information. Furthermore, as part of the interaction with the TAPI interface in redirecting the data to a particular transport, an identifier is returned that can be used by the application to access the data through the desired transport.

The proxy client component relieves the application from knowing anything more than the TAPI interface and exposes potential media as TAPI line devices thereby

simplifying application development. Furthermore, simplified device drivers for connection-oriented media can be written to the integrating component without implementing an API, such as TAPI, for providing connection creation and control to an application.

These and other objects and features of the present invention will become more fully apparent from the following description and appended claims, or may be learned by the practice of the invention as set forth hereinafter.

BRIEF DESCRIPTION OF THE DRAWINGS

In order that the manner in which the above-recited and other advantages and objects of the invention are obtained, a more particular description of the invention briefly described above will be rendered by reference to specific embodiments thereof which are illustrated in the appended drawings. Understanding that these drawings depict only typical embodiments of the invention and are not therefore to be considered limiting of its scope, the invention will be described and explained with additional specificity and detail through the use of the accompanying drawings in which:

Figure 1 is a block diagram showing the environment and interfacing of network card device drivers and transport protocol drivers as existing in the prior art;

Figure 2 is a block diagram showing the introduction of an integrating component that reduces the development complexity of network card device drivers and transport protocol drivers that constitutes the current environment of the present invention;

Figure 3 is a general block diagram showing connection-oriented hardware with accompanying connection-oriented device driver as it fits into the scheme shown in Figure 2;

Figure 4 is a more specific block diagram functionally equivalent to that of Figure 3 wherein specific elements of the NT operating environment are introduced;

Figure 5 is a block diagram of an exemplary system for implementing the present invention that includes a general purpose computing device in the form of a conventional personal computer;

Figure 6 is a block diagram showing an enhanced integrating component according to the present invention corresponding to Figure 3 and having a connection manager component separate from the device driver;

Figure 7 is a block diagram showing an enhanced integrating component according to the present invention corresponding to Figure 3 wherein the connection management

WORKMAN, NYDEGGER & SEELEY
A PROFESSIONAL CORPORATION
ATTORNEYS AT LAW
1000 EAGLE GATE TOWER
60 EAST SOUTH TEMPLE
SALT LAKE CITY, UTAH 84111

functionality is contained within the device driver but uses the same connection manager interface of the integrating component;

Figure 8 is a flow chart showing the processing steps for initialization of the various components shown in Figure 6;

Figure 9 is a flow chart showing the processing steps taken by the different components of Figure 6 in order make an outgoing call or connection so that data may be transmitted over the connection through a data channel;

Figure 10 is a flow chart showing the processing steps taken by the components shown in Figure 6 for receiving an incoming call or connection and processing the same;

Figure 11 is a flow chart showing the processing steps taken by the components in Figure 6 wherein a client closes an existing connection as would occur when a client closes a connection that it had previously made;

Figure 12 is a flow chart showing the processing steps taken by the components in Figure 6 wherein the connection manager receives an end connection indication such as would occur when the other end of a connection terminates the connection.

Figure 13 is a block diagram similar to that shown in Figure 6 showing the addition of a proxy client component that will expose TAPI line devices to an application according to one aspect of the present invention.

Figure 14 is a flow chart showing the processing steps for initialization of the proxy client component shown in the block diagram of Figure 13.

Figure 15 is a flow chart showing the processing steps for initialization of the data transport drivers in order to utilize the services of the proxy client component.

Figure 16 is a flow chart showing the processing steps initiated by an application in order to open a TAPI line device using the proxy client component to interact with the integrating component.

WORKMAN, NYDEGGER & SEELEY
A PROFESSIONAL CORPORATION
ATTORNEYS AT LAW
1000 EAGLE GATE TOWER
60 EAST SOUTH TEMPLE
SALT LAKE CITY, UTAH 84111

Figure 17 is a flow chart showing the processing steps initiated by an application in order to register for an incoming connection through the proxy client component.

Figure 18 is a flow chart showing the processing steps taken in order to redirect the data to a designated data transport using the proxy client component.

Figures 19A-19C are block diagrams that together show the operation of setting up a connection using the proxy client component and then redirecting the data to two different data transports at different points in time by applying the processing steps shown in Figure 18. Specifically, Figure 19A shows the initial set up, Figure 19B shows a data redirection to a first data transport, and Figure 19C shows a data redirection to a second data transport.

WORKMAN, NYDEGGER & SEELEY
A PROFESSIONAL CORPORATION
ATTORNEYS AT LAW
1000 EAGLE GATE TOWER
60 EAST SOUTH TEMPLE
SALT LAKE CITY, UTAH 84111

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

As used herein, the term "software component" refers to any set of executable instructions separately cognisable to an operating system that manages a computer system. Examples would include separate processes or threads, application programs, system device drivers, and any other such operating system entity.

As used herein, the term "communications network" is to be interpreted broadly and includes, but is not limited to, local area networks, telecommunications networks, wide area networks, modem connections, etc. Typically, a communications network will comprise a physical component or physical connection that is made up of the wiring, interface cards and other hardware combined with a specified communications protocol to transmit information from one physical connection to another. The physical network may also be referred to as a specific "media" wherein different media types require different hardware adapters. Furthermore, "signaling protocols" are used to communicate on a particular media type for creation and management of a connection.

As used herein, the term "driver" refers to software driver programs running in a privileged system environment and that interacts with an I/O subsystem as part of an operating system architecture. Such drivers are distinguishable from application programs and other software. A simplified connection-oriented device driver or driver refers to a driver that is smaller in size, less complex, easier to make, or otherwise benefitted over an ordinary driver due to the advantages disclosed herein.

As used herein, the term "direct call linkage" refers to a function call interface. The actual address resolution may be done at compile time through traditional linkers or may be done dynamically by system components when using such entities as dynamic link libraries or export libraries. An invocation session is created when a subroutine is initially called and ends when that particular subroutine ends. An Application Programming Interface (API) is a set of subroutines provided by one software component so that relevant services may be

uniformly accessed. The term "interface" as used in connection with a presently preferred embodiment of the present invention is a direct call linkage between different subroutines provided by the different components. These are typically centered around the integrating component that will provide a number of programming interfaces (*e.g.*, connection interface, connection-oriented data transport interface, etc.) that other components may directly call. Furthermore, as part of these interfaces, a component may be required to provide entry points into subroutines that it will provide so that execution flow may pass from one entity to the other.

As used herein, the term "messaging" refers to sending uniform message blocks between software components in order to communicate therebetween. One example would be the I/O Request Packet (IRP) structure used in the Windows NT I/O system to communicate between different drivers.

Figure 5 and the following discussion are intended to provide a brief, general description of a suitable computing environment in which the invention may be implemented. Although not required, the invention will be described in the general context of computer-executable instructions, such as program modules, being executed by a personal computer. Generally, program modules include routines, programs, objects, components, data structures, etc. that perform particular tasks or implement particular abstract data types. Moreover, those skilled in the art will appreciate that the invention may be practiced with other computer system configurations, including hand-held devices, multi-processor systems, microprocessor-based or programmable consumer electronics, network PCs, minicomputers, mainframe computers, and the like. The invention may also be practiced in distributed computing environments where tasks are performed by remote processing devices that are linked through a communications network. In a distributed computing environment, program modules may be located in both local and remote memory storage devices.

With reference to Figure 5, an exemplary system for implementing the invention includes a general purpose computing device in the form of a conventional personal computer 120, including a processing unit 121, a system memory 122, and a system bus 123 that couples various system components including the system memory to the processing unit 121. The system bus 123 may be any of several types of bus structures including a memory bus or memory controller, a peripheral bus, and a local bus using any of a variety of bus architectures. The system memory includes read only memory (ROM) 124 and random access memory (RAM) 125. A basic input/output system 126 (BIOS), containing the basic routines that helps to transfer information between elements within the personal computer 120, such as during start-up, is stored in ROM 124. The personal computer 120 further includes a hard disk drive 127 for reading from and writing to a hard disk, not shown, a magnetic disk drive 128 for reading from or writing to a removable magnetic disk 129, and an optical disk drive 130 for reading from or writing to removable optical disk 131 such as a CD ROM or other optical media. The hard disk drive 127, magnetic disk drive 128, and optical disk drive 130 are connected to the system bus 123 by a hard disk drive interface 132, a magnetic disk drive-interface 133, and an optical drive interface 134, respectively. The drives and their associated computer-readable media provide nonvolatile storage of computer readable instructions, data structures, program modules and other data for the personal computer 120. Although the exemplary environment described herein employs a hard disk, a removable magnetic disk 129 and a removable optical disk 131, it should be appreciated by those skilled in the art that other types of computer readable media which can store data that is accessible by a computer, such as magnetic cassettes, flash memory cards, digital video disks, Bernoulli cartridges, random access memories (RAMs), read only memories (ROM), and the like, may also be used in the exemplary operating environment. Furthermore, computer readable media includes, but is not limited to, carrier waves over wire

WORKMAN, NYDEGGER & SEELEY
A PROFESSIONAL CORPORATION
ATTORNEYS AT LAW
1000 EAGLE GATE TOWER
60 EAST SOUTH TEMPLE
SALT LAKE CITY, UTAH 84111

or wireless having a data signal embodied therein that represents instructions for use by the computer.

A number of program modules may be stored on the hard disk, magnetic disk 129, optical disk 131, ROM 124 or RAM 125, including an operating system 135, one or more application programs 136, other program modules 137, and program data 138. A user may enter commands and information into the personal computer 120 through input devices such as a keyboard 140 and pointing device 142. Other input devices (not shown) may include a microphone, joystick, game pad, satellite dish, scanner, or the like. These and other input devices are often connected to the processing unit 121 through a serial port interface 146 that is coupled to the system bus 123, but may be connected by other interfaces, such as a parallel port, game port or a universal serial bus (USB). A monitor 147 or other type of display device is also connected to the system bus 123 via an interface, such as a video adapter 148. In addition to the monitor, personal computers typically include other peripheral output devices (not shown), such as speakers and printers.

The personal computer 120 may operate in a networked environment using logical connections to one or more remote computers, such as a remote computer 149. The remote computer 149 may be another personal computer, a server, a router, a network PC, a peer device or other common network node, and typically includes many or all of the elements described above relative to the personal computer 120, although only a memory storage device 150 has been illustrated in Figure 5. The logical connections depicted in Figure 5 include a local area network (LAN) 151 and a wide area network (WAN) 152. Such networking environments are commonplace in offices enterprise-wide computer networks, intranets and the Internet. Furthermore, computer readable instructions may be transmitted over such networking environments such that they represent another form of computer readable media.

WORKMAN, NYDEGGER & SEELEY
A PROFESSIONAL CORPORATION
ATTORNEYS AT LAW
1000 EAGLE GATE TOWER
60 EAST SOUTH TEMPLE
SALT LAKE CITY, UTAH 84111

When used in a LAN networking environment, the personal computer 120 is connected to the local network 151 through a network or adapter 153. When used in a WAN networking environment, the personal computer 120 typically includes a modem 154 or other means, such as an ATM adapter, for establishing communications over the wide area network 152, such as the Internet. The modem 154, which may be internal or external, is connected to the system bus 123 via the serial port interface 146. In a networked environment, program modules depicted relative to the personal computer 120, or portions thereof, may be stored in the remote memory storage device. It will be appreciated that the network connections shown are exemplary and other means of establishing a communications link between the computers may be used.

Referring now to Figure 6, a logical diagram presenting an integrating component that provides an abstracted connection interface and an additional connection manager component according to the present invention is shown. Figure 6 corresponds functionally in many respects as Figure 3 with an important benefit being that the connection-oriented device driver may be written in more simplified fashion. In other words, the connection management ability is handled by the connection manager component and not the device driver. Furthermore, the connection interface with upper layers of code is standardized at the integrating component relieving this additional burden from the device driver as well.

The connection-oriented hardware adapter 156 provides access to a particular connection-oriented media. For example, the connection-oriented hardware adapter 156 may be an ATM adapter that will allow connection onto an ATM network. Other kinds of connection-oriented communications networks include Integrated Services Data Network (ISDN), Plain Old Telephone Service (POTS), etc.

The connection-oriented hardware adapter 156 is controlled and managed by a simplified connection-oriented device driver 158 as indicated by the arrow 160. This simplified connection-oriented device driver will require less effort to develop since

WORKMAN, NYDEGGER & SEELEY
A PROFESSIONAL CORPORATION
ATTORNEYS AT LAW
1000 EAGLE GATE TOWER
60 EAST SOUTH TEMPLE
SALT LAKE CITY, UTAH 84111

connection management capability has been removed from the device driver and it will no longer have to provide a connection interface. The integrating component 162 has been enhanced by providing an abstracted connection interface and a call management interface that will be explained hereafter. As before, the simplified connection-oriented device driver 158 will interact with the integrating component 162 through the device driver interface 164 as indicated by arrow 166. The simplified connection-oriented device driver 158 will be written to a standardized device driver interface 164 that will allow all necessary connection functions to be carried out by the device driver as part of controlling the underlying hardware.

A plurality of connection manager components or connection managers 168 will also interact with the integrating component 162 over a connection management interface 170 as indicated by the arrow 172. A connection manager may be developed independently of the simplified connection-oriented device driver 158 so that a common connection manager may be used with a variety of different device drivers 158. This would allow the provider of the integrating component 162 to also provide one or more connection managers as part of the plurality of connection managers 168 so that a manufacturer of a connection-oriented hardware adapter 156 need only provide the simplified connection-oriented device driver 158 and need not be concerned about providing connection functionality nor a separate connection interface as part of the device driver code.

Each connection manager of the plurality of connection managers 168 is capable of supporting or providing one or more address families. As used herein, the term "address families" refer to signaling protocols used to make and manage connections on a particular media type. These signaling protocols will be used to control an adapter as well as communicate across the specific media type in order to create and maintain connections.

In some instances, a connection manager may support multiple address families. For example, on an ATM network an address family for the Q2931 protocol may be supported

WORKMAN, NYDEGGER & SEELEY
A PROFESSIONAL CORPORATION
ATTORNEYS AT LAW
1000 EAGLE GATE TOWER
60 EAST SOUTH TEMPLE
SALT LAKE CITY, UTAH 84111

for a 3.1 version or a newer 4.0 version. The corresponding connection manager could provide either or both of these address families to be made available to clients. In this manner, a legacy client that is unaware of the newer signaling protocol version may compatibly bind with the older version. The connection interface provided by the integrating component will allow a client to query the different address families that are registered by a connection manager so that the client may choose an address family that is known to the client.

An application 174 will eventually communicate with the integrating component 162 for connection purposes through the connection interface 176 as indicated by arrow 178. Arrow 178 may represent a number of different protocols and components in order to make and manage such connection depending on the configuration of the system and what protocols are installed. In any case, direct communication with the integrating component is done through a client 180 that is shown in outline form in Figure 6. The client is any component that directly interacts with the integrating component over the connection interface 176 regardless of what other functions the client may perform. In other words with respect to the connection interface 176 anything, whether it be the application itself 174 or some other component, such as a data transport protocol, device driver, etc., will be considered the client 180 and will operate in the same manner over the connection interface 176.

With respect to actually sending data over a created connection, a connection-oriented transport 182 may be used to communicate with the connection-oriented data transport interface as indicated by arrow 186. Such a connection-oriented data transport 182 would indicate a particular connection by reference to a connection identifier that identifies the particular connection over which data may be transported. Further, the connection-oriented data transport 182 will communicate with an application program 174 as indicated by the arrow 188. Again, the arrow 188 may represent a number of different components in

order to make the communication between the application and the connection-oriented data transport 182.

Note that the integrating component 162 with its connection interface 176 and the connection-oriented data transport interface 184 may have a single client that will operate both interfaces. Furthermore, a connectionless data transport interface (not shown) may also make up part of the integrating component 162 as explained previously. The logical diagram in Figure 6 is simplified in order to present the aspects of the present invention and those skilled in the art will quickly observe that many other different kinds of functionality may be incorporated into the integrating component 162.

Referring now to Figure 7, a logical diagram presenting an integrating component that provides an abstracted connection interface and connection manager interface according to the present invention is shown. Figure 7 corresponds functionally in many respects as Figure 3 with an important benefit being that the connection interface is abstracted from the details of how the connection is provided by a particular media type. The device driver is relieved from providing an interface and will provide the connection functionality through the connection manager interface, also provided by the integrating component.

Note that the connection management interface is substantially the same in both Figure 6 and Figure 7 with only minor differences in order to determine whether the connection management functionality is included as part of the device driver or is an entirely separate component. In either case, the connection management will occur in the same manner as explained hereafter

The connection-oriented hardware adapter 190 provides access to a particular connection-oriented media. For example, the connection-oriented hardware adapter 190 may be an ATM adapter that will allow connection onto an ATM network. Other kinds of connection-oriented communications networks include Integrated Services Data Network (ISDN), Plain Old Telephone Service (POTS), etc.

WORKMAN, NYDEGGER & SEELEY
A PROFESSIONAL CORPORATION
ATTORNEYS AT LAW
1000 EAGLE GATE TOWER
60 EAST SOUTH TEMPLE
SALT LAKE CITY, UTAH 84111

The connection-oriented hardware adapter 190 is controlled and managed by a connection-oriented device driver 192 as indicated by the arrow 193. This connection-oriented device driver will require less effort to develop since it will no longer have to provide a connection interface. The integrating component 194 has been enhanced by providing an abstracted connection interface and a call management interface that will be explained hereafter. As before, the connection-oriented device driver 190 will interact with the integrating component 194 through the device driver interface 196 as indicated by arrow 198. The connection-oriented device driver 190 will be written to the standardized device driver interface 196 that will allow all necessary connection functions requiring hardware control to be carried out by the device driver as part of controlling the underlying hardware.

A connection manager portion 200 of the connection-oriented device driver 192 will also interact with the integrating component 194 over a connection management interface 202 as indicated by the arrow 204.

The connection manager portion 200 of the connection-oriented device driver 192 is capable of supporting or providing one or more address families. As explained previously, the term "address families" refer to signaling protocols used to make and manage connections on a particular media type. These signaling protocols will be used to control an adapter as well as communicate across the specific media type in order to create and maintain connections.

In some instances, a connection manager may support multiple address families. For example, on an ATM network an address family for the Q2931 protocol may be supported for a 3.1 version or a newer 4.0 version. The corresponding connection manager could provide either or both of these address families to be made available to clients. In this manner, a legacy client that is unaware of the newer signaling protocol version may compatibly bind with the older version. The connection interface provided by the integrating component will allow a client to query the different address families that are registered by

WORKMAN, NYDEGGER & SEELEY
A PROFESSIONAL CORPORATION
ATTORNEYS AT LAW
1000 EAGLE GATE TOWER
60 EAST SOUTH TEMPLE
SALT LAKE CITY, UTAH 84111

a connection manager so that the client may choose an address family that is known to the client.

An application 206 will eventually communicate with the integrating component 194 for connection purposes through the connection interface 208 as indicated by arrow 210. Arrow 210 may represent a number of different protocols and components in order to make and manage such connection depending on the configuration of the system and what protocols are installed. In any case, direct communication with the integrating component is done through a client 212 that is shown in outline form in Figure 7. The client is any component that directly interacts with the integrating component over the connection interface 208 regardless of what other functions the client may perform. In other words with respect to the connection interface 208 anything, whether it be the application itself 206 or some other component, such as a data transport protocol, device driver, etc., will be considered the client 212 and will operate in the same manner over the connection interface 208.

With respect to actually sending data over a created connection, a connection-oriented transport 214 may be used to communicate with the connection-oriented data transport interface as indicated by arrow 218. Such a connection-oriented data transport 214 would indicate a particular connection by reference to a connection identifier that identifies the particular connection over which data may be transported. Further, the connection-oriented data transport 214 will communicate with an application program 206 as indicated by the arrow 220. Again, the arrow 220 may represent a number of different components in order to make the communication between the application and the connection-oriented data transport 214.

Note that the integrating component 194 with its connection interface 208 and the connection-oriented data transport interface 216 may have a single client that will operate both interfaces. Furthermore, a connectionless data transport interface (not shown) may also

WORKMAN, NYDEGGER & SEELEY
A PROFESSIONAL CORPORATION
ATTORNEYS AT LAW
1000 EAGLE GATE TOWER
60 EAST SOUTH TEMPLE
SALT LAKE CITY, UTAH 84111

make up part of the integrating component 194 as explained previously. The logical diagram in Figure 7 is simplified in order to present the aspects of the present invention and those skilled in the art will quickly observe that many other different kinds of functionality may be incorporated into the integrating component 194.

Referring now to Figure 8, a flow chart showing the processing steps for binding and initialization of the different components is shown. As used herein, the term "binding" refers to the process of making a logical association between different components. Furthermore, components that are "bound" together may form a data channel or a connection control channel. The bar 230 lists each of the different components other than the integrating component and the corresponding steps below the titled component indicates steps pertinent to that respective component.

With respects to the connection-oriented device driver, such as the connection-oriented device driver 158 of Figure 6, the device driver code will register the device driver with the integrating component at step 232. This registration process is designed to make the integrating component aware of the device driver and its various capabilities and settings. Furthermore, certain entry points into the device driver code will also be made known to the integrating component so that execution flow and control may be passed from the integrating component to the device driver by a simple function call mechanism. In this manner, certain entry points, called handler functions, are registered with the integrating component. Other components will communicate with the integrating component by calling defined functions of the integrating component API. An "interface" on the integrating component comprises the API subroutines that will be called for the interface functionality as well as the provided handlers by the different components. Those skilled in the art will note that other mechanisms, such as message passing or packet passing may also be used to communicate between the different components.

WORKMAN, NYDEGGER & SEELEY
A PROFESSIONAL CORPORATION
ATTORNEYS AT LAW
1000 EAGLE GATE TOWER
60 EAST SOUTH TEMPLE
SALT LAKE CITY, UTAH 84111

At step 234, the device driver code will initialize the adapter and take all necessary steps in order to have the adapter and the device driver ready for making or receiving connections and otherwise operating the adapter to send data over the particular media type.

At step 236, the connection manager will register itself as a protocol with the integrating component. Components that are registered with the integrating component that are not device drivers are considered protocol drivers and would include components such as the connection manager, a client of the integrating component for making connections or transmitting data, or a data transport protocol driver (whether connection-oriented or connectionless), etc. At step 238, the connection manager will receive information from the integrating component regarding available adapters, their media type, and connectivity information. Such information may be queried, received from the integrating component, accessed from the registry, etc. according to the particular implementation.

Binding by the connection manager with the various adapters begins at step 240. At this point, the connection manager will designate the various adapters to which it will bind or to which it otherwise pertains. For example, an ATM connection manager would designate the appropriate ATM hardware adapters and select them for binding. Further resource may be made to the registry information in making this determination or it may be done automatically without explicit configuration on the part of a user. Those skilled in the art will recognize that a number of different options may be used to accomplish an appropriate binding.

The address family or families that the connection manager supports for a particular adapter is registered with the integrating component at step 242. The address family indicates the media type and signaling protocol that a particular connection manager will support. An address family will be used by a client for making and using a connection over the particular media type. For example, for an ATM connection manager the address family would be related to Q2931 standard. Such address families will be available to various

WORKMAN, NYDEGGER & SEELEY
A PROFESSIONAL CORPORATION
ATTORNEYS AT LAW
1000 EAGLE GATE TOWER
60 EAST SOUTH TEMPLE
SALT LAKE CITY, UTAH 84111

clients who will "open" an address family in order to make the binding or association between the client and the connection manager complete.

Separately, the connection manager and the client will also bind or associate with the particular hardware connection-oriented adapter and corresponding device driver for a particular media type. In this manner, a three-way association between the various parts (*i.e.*, client, connection manager, and device driver) are made through the integrating component. The integrating component will also maintain many of the common data structures used by the different I/O subsystem parts.

An optional step 244 may be taken for creating a virtual connection depending upon media type. A virtual connection that is not associated with an actual connection between clients would be used by the connection manager for interacting with the device driver controlling the connection-oriented hardware adapter. For example, ATM adapters may require such a "private" virtual connection in order to asses quality of service availability on the ATM network. Also, in the ATM case, the private virtual connection is created to allow the device driver to exchange signaling messages with the ATM network.

Finally, the connection manger registration and initialization is completed when at step 246 the binding with the adapters is completed. The integrating component will match such address family registration information with the parameters supplied by a client when attempting to "open" an address family so that the association or binding may be made correctly.

With respect to the client, it will register itself as a protocol at step 248 so that the integrating component knows of its existence, capabilities, and various entry points in order to operate with the established interfaces. A client may use a number of different integrating component interfaces in order to accomplish its desired function. For example, the client may operate with solely the connection interface in order to make connections and pass identifiers created connections to an application or some other layer further up the protocol

WORKMAN, NYDEGGER & SEELEY
A PROFESSIONAL CORPORATION
ATTORNEYS AT LAW
1000 EAGLE GATE TOWER
60 EAST SOUTH TEMPLE
SALT LAKE CITY, UTAH 84111

stack for use in data transmission over the connection. Alternatively, a client may both make the connection as well as send data over the connection-oriented data transport interface using the connection identifier previously received. Depending on other interfaces provided by the integrating component, such as a connectionless interface, any client may interface thereto in addition to the connection interface. Finally, as explained in more detail hereafter, an application may interact with a proxy client component that will relieve an application from knowing the details of the connection interface.

At step 250, a new client will receive integrating component status information from the integrating component in some fashion, whether by a data structure parameter into an entry point or querying or other communication mechanism known in the art. With this information, the client will do an adapter binding at step 252 in order to make a binding or an association with the correct hardware media adapter and corresponding device driver. In this fashion, both the connection manager and the client have been bound or associated with the same adapter/device driver. Next, at step 254, the client will "open" an address family or otherwise make the association or binding between the client and the connection manager. Again, this is done is done using the API calls pertinent to the connection interface provided by the integrating component in one embodiment of the present invention. As explained previously, those skilled in the art will note that other forms are known for communicating between components and integrating component such as messaging, etc.

Finally, at step 256, the client will register a service access point in order to receive notification of incoming calls destined to be made into connections. When a service access point is registered, a particular connection manager will receive the incoming call and will dispatch it to the client having the appropriate service access point registered so that the actual connection may be made as will be explained in more detail hereafter.

Referring now to Figure 9, a flow chart showing the processing steps taken by the various components in order to create a connection over a particular media type is shown.

Note that these steps are taken only to make the connection and prepare the system for sending or receiving data over the previously connection. Data transmission may be done by the client or the connection identifier may be passed to a different component client to the integrating component.

The bar 258 over Figure 9 indicates the different components shown in Figure 6 that may be operating at any given point in time. The processing steps indicated below the signified component (*e.g.*, client, integrating component, connection manager, or device driver) will be where the designated functionality is actually performed. Transitions between the different components may occur in a variety of manners and by making function calls through the integrating component. Note also that in almost every instance, communication will occur through the integrating component though not necessarily shown. For example, going from step 264 to step 266 occurs between the connection manager and the device driver as shown in Figure 9, but the actual flow of execution will have passed through the integrating component. This is done in order to focus on the functional elements performed by the differing components.

When a client desires to make a connection to a destination node, it will begin the process of creating a virtual connection at step 260. A virtual connection is first created and then activated. Furthermore, since a virtual connection may be created by either a client (for outgoing calls or connections) or a connection manager (for receiving calls or connections) only the entity that created the virtual connection may later destroy the virtual connection. This rule is enforced by the integrating component which will actually perform the allocation of the data structures and track the status of the virtual connection.

At step 262, the integrating component will actually make the virtual connection data structure and an identifier for that virtual connection will be received by the connection manager at step 264 so that appropriate connection manager processing may occur. Next, the identifier for the virtual connection will be received by the connection-oriented device

WORKMAN, NYDEGGER & SEELEY
A PROFESSIONAL CORPORATION
ATTORNEYS AT LAW
1000 EAGLE GATE TOWER
60 EAST SOUTH TEMPLE
SALT LAKE CITY, UTAH 84111

driver at step 266 so that device driver may also perform appropriate connection processing. In one preferred embodiment, the reception of the identifier is received by having the integrating component call a "handler" routine whose entry point had been previously provided during the registration step for the respective component (*e.g.*, connection manager, device driver).

At step 268, execution control will be back to the client where the client will initiate the connection using the virtual connection identifier received from the integrating component that represents the virtual connection. The connection manager will then activate the virtual connection at step 270 by issuing the appropriate commands to the device driver. At this point, the device driver will have the adapter make the connection at step 272 and receive a connection made indication at step 274. Completion of the connection is then indicated at step 276 for the connection manager which in turn indicates through the integrating component to the client that connection was made.

At step 278, the client receive notification of the connection being made with status information and a virtual connection identifier that may be used in order to communicate over the connection. Finally, the client may begin data transmission and received over the virtual connection using the virtual connection identifier and the connection-oriented interface of the integrating component. In other words, the connection-oriented data transport interface subroutine calls or other communication mechanism will require the virtual connection identifier as part of the parameter list so that the correct connection will be used in a system having many simultaneous virtual connections in operation. Additionally, for some media types, such as ATM, many different virtual connections may be made simultaneously over the same hardware and having a virtual connection identifier allows each individual connection to be separately used and be logically separated.

Referring now to Figure 10, a flow chart showing the processing steps performed by the various system components that occur when receiving notification of an incoming call

WORKMAN, NYDEGGER & SEELEY
A PROFESSIONAL CORPORATION
ATTORNEYS AT LAW
1000 EAGLE GATE TOWER
60 EAST SOUTH TEMPLE
SALT LAKE CITY, UTAH 84111

in order to make a connection and to begin communication over that connection is shown. Again, the bar 282 indicating which of the various components has control or performs each processing step is shown above the respective steps.

Initially, processing begins with the connection manager receiving notification of an incoming call at step 284. This notification will be transmitted from the hardware and device driver as they interact together as well as further interaction with the integrating component and operating system.

A virtual connection is created by the connection manager at step 286. This is done through a function call invocation to the integrating component which receives control and actually makes the virtual connection at step 288. The integrating component will create the data structures associated with the virtual connection and manage the logical lifetime of the virtual connection.

Existence of the virtual connection and its identifier is next transmitted to both the client at step 290 and to the device driver at step 292 from the integrating component. In other words, both the client and the device driver will receive notification of the virtual connection creation and the virtual connection identifier and will do all associated initial processing. This results in a valid virtual connection created with all the active parties to the connection (*i.e.*, client, connection manager, and device driver) having the identifier and having done all initial processing.

Next, at step 294, the connection manager will activate the virtual connection in order to receive the incoming call or connection. This is done by having the media adapter make the connection at step 296 over the media.

The device driver will receive notification that the connection has been made at 298 before turning control back to the connection manager at step 300 where the virtual connection will be completed at step 300. Next, the notification of the incoming and completed connection will be dispatched to the client for handling at step 302. This is done

WORKMAN, NYDEGGER & SEELEY
A PROFESSIONAL CORPORATION
ATTORNEYS AT LAW
1000 EAGLE GATE TOWER
60 EAST SOUTH TEMPLE
SALT LAKE CITY, UTAH 84111

by communicating this to the integrating component which will dispatch the connection notification to the appropriate client based on ownership of a previously registered service access point. The connection characteristics will be matched with a group of registered service access points at the integrating component in order to determine which client should receive the connection. Finally, the client will handle the incoming connection at step 306 and make whatever housekeeping calls are necessary to recognize the incoming connection.

The connection manager will then receive notification that the connection is complete at step 308, meaning that the client is aware of the connection and waiting for final status. The connection manager will make the completion of the connection in preparation for use by the client of step 308 before control is passed to the client.

At step 310, the client receives notification that the connection is made with status regarding that connection that includes but is not limited to the virtual connection identifier and status that the connection is made and ready to receive data. This virtual connection identifier is used for communicating over the connection at step 312 whether to receive data or to send data. Note also that the client may pass the virtual connection identifier to another software component that may in turn use it for communicating over the connection-oriented data transport interface of the integrating component. In either case, the connection is ready to handle data.

Referring now to Figure 11, a flow chart showing the processing steps for ending a connection as initiated by a client is shown. The bar 314 has listed therein the different system components and below each component name would be the processing steps performed by that respective component.

The client begins at step 316 by closing the connection by issuing the appropriate integrating component subroutine call. The integrating component will take processing control at step 318 and mark the identified virtual connection as closing and pass control to the connection manager.

At step 320, the connection manager will communicate over the connection signaling messages for releasing the connection. These messages are sent to the destination node of the connection so that the connection may be terminated at that end as well.

At step 322, the connection manager will deactivate the virtual connection so that it no longer is in use. Control passes to the device driver, where at step 324 the device driver will instruct the adapter to physically end the connection over the media. Next, control flows back to the connection manager where the connection completion is finished as far as the connection manager is concerned at step 326.

Control then passes to the integrating component where the virtual connection is marked as closed at step 328. The virtual connection will remain in existence until explicitly deleted but will be closed until activated at some future point in time.

Finally, the client receives notification at step 330 that the connection has been closed and completes any necessary cleanup to finish the closing of the connection. A virtual connection may be deleted at step 332 if necessary or desirable. Step 332 is optional since it may also be desirable to leave the connection in existence so that it may be activated and used at a later time.

Referring now to Figure 12, a flow chart showing the processing steps taken to end a connection wherein the ending of the connection is initiated by the other side of the connection is shown. The bar 334 containing the names of the various components of the system is used to indicate which particular component has processing control for each of the respective processing steps that are discussed.

Initially, the connection manager will receive end connection signals over the connection at step 336. How this is done will depend on the signaling mechanisms of the particular media types but will be known to the connection manager for that particular media type. Once the termination has been made known to the connection manager, this

information is dispatched by the connection manager at step 338 to the integrating component for distribution to other associated components.

At step 340, the integrating component will receive the dispatch notification from the connection manager and pass the closing connection notification to the client. The client receives this at step 342 and begins relevant processing to close the connection. Part of this processing will entail initiating a close connection command through a subroutine call at step 344. Again, the integrating component receives the close connection call and will mark the virtual connection as closing at step 346.

The responsibility for closing the connection will then be passed to the connection manager which will deactivate the virtual connection at step 348. As part of this deactivation of the virtual connection, instructions will be passed to the device driver so that the device driver will have the adapter physically end the connection as indicated at step 350. Once the device driver has ended the connection, control passes again to the connection manager for any post processing necessary to finish ending the connection. Control is then passed to the integrating component so that notification may be made to the client and the integrating component will know that the connection manager has ended the connection.

At step 354, the integrating component receives control and marks the virtual connection as closed at step 350 before passing notification on to the client. At step 356, the client receives notification that the connection has been closed and will take any processing steps necessary to so manage a closed connection. Finally, control will pass back to the connection manager where, at step 358, the virtual connection may optionally be deleted. Note that this is an optional step and in some instances the connection manager may or may not desire to end the connection.

Referring now to Figure 13, a block diagram showing an integrating component having a connection interface and capable of handling simplified connection-oriented device drivers as explained previously is shown. Additionally, Figure 13 shows a special proxy

client component that allows the functionality of the I/O subsystem to be exposed as more familiar TAPI line devices. This is advantageous because application developers will write to TAPI and will not need to learn or otherwise write code interacting with the connection interface of the integrating component. After outlining the pieces of the block diagram, an explanation of how the proxy client component works and achieves its purpose is explained in more detail.

The integrating component 360 interacts with a plurality of simplified connection-oriented device drivers 362 as indicated by arrow 364 through the device driver interface 365. The plurality of simplified connection-oriented device drivers 362 will control corresponding plurality of connection-oriented hardware 366 as indicated by arrow 368. A computer system, such as that shown in Figure 5, may have only one connection-oriented hardware device or there may be several depending on the system's configuration. Also, as explained previously, a plurality of connection managers 370 will interact as indicated by arrow 372 with a connection manager interface 374 that is part of the integrating component 360.

On the upper edge of the integrating component 360, an application 376 interacts with the plurality of connection-oriented data transports 378 as indicated by arrow 380. Note that in many instances an application will only need a single data transport since it may be directed to a very specific type of data, but increasingly, applications will deal with multiple kinds of data formats over the connection-oriented transports 378. In order to fully make the data path, the connection-oriented data transports 378 interact with the connection-oriented data transport interface found on the integrating component 360 as indicated by arrow 384.

In this instance, the application 376 is written to interact with a specific interface for connection-oriented media known as the Telephony Application Programming Interface or TAPI 386 as indicated by arrow 388. The proxy client component 390 will implement the TAPI commands and respond to them as indicated by arrow 392. In this fashion, an

application may gain the benefits of the improved connection-oriented I/O subsystem without having to learn a new set of programming interfaces. The proxy client component 390 is able to expose the components of the I/O subsystem as TAPI line devices. Its ability to query the integrating component 360 and make all the necessary "bindings" as well as interact appropriately is achieved by communicating with the integrating component according to three main interfaces. These interfaces are the connection-oriented data transport interface 382 as indicated by arrow 394, the connection interface 395 as indicated by arrow 396 and 396 the connection manager interface 374 as indicated by arrow 398. Details of how the proxy client component 390 is able to achieve these functions will be explained in more detail hereafter.

The ability of the proxy client component to effectively provide information in TAPI compatible format lies in its dual nature as a data transport and a connection manager. Other connection managers 370 will "see" and interact with the proxy client component 390 as a regular data transport. Similarly, other data transports 378 will "see" and interact with the proxy client component 390 as a regular connection manager. Finally, the proxy client component 390 will present TAPI line devices to applications so that the applications are unaware of what is happening underneath in terms of the integrating component, data transports, connection managers, etc. Also, the proxy client component 390 will also make and maintain connections over the connection interface 395.

The complexity of the I/O subsystem is effectively hidden from the application 376 so that applications can be easily programmed to a known and existing protocol, such as TAPI 386. Those skilled in the art will recognize that other interfaces known at the application level besides TAPI could be chosen and TAPI is used by way of example and not limitation. Because of the integrating component, the device drivers can also be developed without any knowledge of what components are ultimately managing the connection interface.

The proxy client component 390 functions as a bridge between the data transports and the connection managers supporting certain address families. The three way association explained previously is made twice with the proxy client component 390 used in each association; as a connection manager in one and as a data transport in the other. The client proxy component 390 will then dynamically assign the data transport to an existing connection based on a TAPI command so that data will flow through the data transport as requested by an application while control information will continue to flow through the proxy client component 390 and TAPI 386. This process is achieved by initializing the proxy client component 390 and the data transports 378 in the proper fashion (Figures 14 and 15), establishing a connection through the proxy client component 390 (Figures 16 and 17), and then redirecting the data flow from the proxy client component 390 to the desired data transport (Figure 18).

Referring now to the flow chart shown in Figure 14, the processing steps for initializing the proxy client component 390 shown in Figure 13 are presented. At step 400, the proxy client component will register itself as a protocol with the integrating component. At step 402, the proxy client component will receive information from the integrating component regarding available adapters, their media type, and connectivity information. Such information may be queried, received from the integrating component, accessed from the registry, etc. according to the particular implementation. Binding by the proxy client component with the various adapters begins at step 404. The initialization has been done so far in the same fashion as would be done for a connection manager.

In order to allow the data transports that ability to bind with the proxy client component, a special proxy address family is registered at step 406 that will be available to all data transports. Here, the proxy client component is acting like a connection manager. All the data transports will then be able to bind to the proxy client component the same way they would bind to any other connection manager without knowing the difference. In this

sense, the proxy client component fully implements the connection manager interface. The proxy client component registration continues when, at step 408, the binding with the adapters is completed.

Next, at step 410, the proxy client component will "open" all the address families registered by the various connection managers in order to make the association or binding between the proxy client component and all the connection managers. At this point, the proxy client component is acting like a regular data transport. The client proxy component will use the connection managers to make and control connections.

At step 412, the proxy client component will query capabilities of the connection managers, underlying media, device drivers, etc. in order to create and expose one or more TAPI line devices that are accessible by an application through TAPI. Note that a virtual connection was not created nor a service access point registered. These events will occur as shown hereafter in response to TAPI commands that are given to the proxy client component.

Referring now to the flow chart shown in Figure 15, the data transports will each register as a protocol at step 414 so that the integrating component knows of their existence, capabilities, and various entry points in order to operate with the established interfaces. At step 416, each data transport will receive integrating component status information from the integrating component in some fashion, whether by a data structure parameter into an entry point or querying or other communication mechanism known in the art. With this information, the client will do an adapter binding at step 418 in order to make a binding or an association with the correct hardware media adapter and corresponding device driver. In this fashion, both the proxy client component and the data transports have been bound or associated with the same adapters/device drivers. Up to this point, the initialization has proceeded as has been explained previously.

At step 420, however, processing diverges somewhat in that each data transport will "open" a special address family previously registered by the proxy client component known as the proxy address family in order make the association or binding between each data transport and the proxy client component. Again, this is done is done using the API calls pertinent to the connection interface provided by the integrating component in one embodiment of the present invention.

With the binding complete between the data transports and the proxy client component, and the connection managers and the proxy client component, the proxy client component can now act as a bridge and dynamically make virtual connections and connections in response to TAPI commands.

Referring now to Figure 16, a flow chart showing the processing steps taken by an application and the proxy client component for opening a TAPI line device and establishing a connection are shown. The bar 422 above the processing steps indicates whether the step is performed by the application or proxy client component.

The application opens a TAPI line device at step 424 whereupon the proxy client component will return status and identifying information for the line device. At step 426, the application will query for the line device characteristics. This will require the proxy client component to represent the capabilities of the connection-oriented hardware and associated connection manager at step 428. The proxy client component may make various calls to the integrating component in order to ascertain all the characteristics of the device including the most current status.

Once the application knows the characteristics and most current status of the TAPI line device, it will use the LineMakeCall command to setup a connection at step 430. This causes the proxy client component to create a virtual connection at step 432. The details of creating a virtual connection were shown in more detail previously in Figure 9, steps 260-266.

WORKMAN, NYDEGGER & SEELEY
A PROFESSIONAL CORPORATION
ATTORNEYS AT LAW
1000 EAGLE GATE TOWER
60 EAST SOUTH TEMPLE
SALT LAKE CITY, UTAH 84111

After the virtual connection is completed, a connection is initiated at step 434. Again, the interaction between the proxy client component as a client to the connection interface for initiating a connection was shown previously in more detail in the discussion of Figure 9, specifically steps 268-278. Finally, the proxy client component will respond with connection status at step 436 which will be received by the application at step 438. At this point, the application is ready to send or receive data over the created connection and will likely use the redirection aspect of the proxy client component in order to achieve this as will be explained in more detail hereafter. Note that a virtual connection was not created until after the TAPI line device was opened and a line make call command was initiated.

Referring now to Figure 17, the processing steps taken by the application and proxy client component in order to notify the application of an incoming call or connection is shown. Again, a bar 440 indicates whether the stream of execution lies with the application or the proxy client component.

Initially, the application will use the appropriate TAPI command for registering the application for an incoming call at step 442. Upon receiving this TAPI command, the proxy client component will register a service access point at step 444. The proxy client does this by acting as a regular data transport and using the appropriate interface of the integrating component.

At this point, the system waits for an incoming connection. The proxy client component will receive notification of the connection as well as an identifier thereof at step 446 from the integrating component. This process is shown in more detail in Figure 10 and the discussion thereof. Finally, the proxy client component will notify the application of the completed connection, including an identifier for controlling that connection, at step 448.

The application receives the identifier and status information for the received connection at step 450 where it will then prepare to send and/or receive data. Note that Figure 17 is simply another way of establishing a connection that differs from the way shown

WORKMAN, NYDEGGER & SEELEY
A PROFESSIONAL CORPORATION
ATTORNEYS AT LAW
1000 EAGLE GATE TOWER
60 EAST SOUTH TEMPLE
SALT LAKE CITY, UTAH 84111

in Figure 16 in that the application does not initiate the connection but is the recipient thereof. Essentially, the application must notify the I/O subsystem that it is ready to receive a connection and if such a connection occurs, the application will then be notified.

Referring now to Figure 18, a flow chart showing the processing steps taken by the application, proxy client component, integrating component, and data transport for redirecting data from one connection to a new virtual connection using a designated data transport while maintaining control information through the original virtual connection is shown. Again, the bar 452 rests above the processing steps to show where a particular processing step is executed.

Initially, the application issues a TAPI lineGetID command with a designated data type at step 454. The proxy client component will request a new virtual connection using the data transport associated with the data type indicated in the lineGetID command at step 456. The proxy client component is able to ascertain the correct data transport due to its knowledge of the various data transports that have been bound thereto by opening the special proxy address family.

When making the new virtual connection, the proxy client component will indicate to the integrating component the existing virtual connection. The knowledge of the existing virtual connection will be interpreted by the integrating component that this is a redirection scenario so that only data will be routed over the new virtual connection. This occurs at step 458 and effectively divides the path by which data and data control information will flow as opposed to connection control information. Again, connection control information will continue to flow by way of the existing virtual connection through the proxy client component to the application while data and data control information will flow by way of the new virtual connection through the specified data transport to the application over a different interface.

As part of the processing for making the new virtual connection, the integrating component will transfer control to the data transport which will in turn be prepared to receive data at step 460. The integrating component will then return the new virtual connection identifier at step 462 to the proxy client component. At step 464, the proxy client component issues an incoming connection notification to the data transport in order to effectuate the redirection of the data and data control information. At step 466, the data transport receives data for the incoming connection (*i.e.* data is now redirected).

Once the redirection is accomplished, the proxy client component will return the new virtual connection identifier to the application at step 468 so that the application may be able to access the data over another interface. Finally, at step 470 the application receives and uses the new virtual connection identifier in order to access the data over the data transport. Note that between the application and the data transport there may be intervening layers of protocol code. In any event, the new virtual connection identifier will be passed down any intervening code so that the correct data may be accessed over the specified data transport.

Referring now to Figure 19A, a I/O subsystem using the proxy client component is shown after having established a connection with a particular connection-oriented hardware device. Note that the control data path flows through the cross-hatched arrows 472-480. This is the state of the system after having executed the processing steps of Figure 16 in order to create a new connection or having executed the processing steps of Figure 17 in registering to receive a incoming connection and having received such a connection.

Figure 19B highlights the data and data control information path after the processing steps shown in Figure 18 have occurred to cause a redirection. This is indicated by cross-hatched arrows 482, 484, 478, and 480. In this particular case, data and data control information is received over the point-to-point protocol (PPP) data transport. The data will come across the PPP data transport in a form that the application can understand. Note that

WORKMAN, NYDEGGER & SEELEY
A PROFESSIONAL CORPORATION
ATTORNEYS AT LAW
1000 EAGLE GATE TOWER
60 EAST SOUTH TEMPLE
SALT LAKE CITY, UTAH 84111

connection control and status information continues to arrive over the original path shown in Figure 19A.

Finally, the application may use the processing steps shown in Figure 18 to redirect the data again in order to read and process a different type of data, in this case unframed voice data. Again, this is indicated by the cross-hatched arrows 486, 488, 478, and 480. Note that in both Figure 19B and 19C that control information will continue to flow in the original path designated in Figure 19A through the TAPI interface to the application. By separating the control information from the data and data control information, a given data transport may be used in a connection-oriented environment without regard to the type of media or hardware device used, or how a connection is created or managed.

The present invention may be embodied in other specific forms without departing from its spirit or essential characteristics. The described embodiments are to be considered in all respects only as illustrated and not restrictive. The scope of the invention is, therefore, indicated by the appended claims rather than by the foregoing description. All changes which come within the meaning and range of equivalency of the claims are to be embraced within their scope.

What is claimed and desired to be secured by United States Letters Patent is:

1. A method for representing to an application the characteristics of an underlying connection-oriented device over a known application-level interface and allowing an application to take advantage of a connection-oriented I/O subsystem having an integrating component over the known application-level interface and without requiring the application programmer to program directly to the integrating component, the method comprising:

representing to an application over a known application-level interface characteristics of an underlying connection-oriented device;

receiving at least one command in the known application-level interface format;

interacting with the integrating component of the connection-oriented I/O subsystem in order to represent underlying connection-oriented device characteristics over a known application-level interface and to execute received commands so that an application may take advantage of the connection-oriented I/O subsystem and use the connection-oriented device using the known application-level interface and without requiring the application programmer to program to a new interface.

2. A method as recited in claim 1 wherein the integrating component has a connection interface for making connections with underlying connection-oriented devices, and a data transport interface for interacting with a data transport component and the interacting with the integrating component comprises the steps of:

having data transport components interact with applications and the data transport interface;

sending, to the integrating component, instructions over the connection interface for directing data and data control information over a specified data transport component; and

WORKMAN, NYDEGGER & SEELEY
A PROFESSIONAL CORPORATION
ATTORNEYS AT LAW
1000 EAGLE GATE TOWER
60 EAST SOUTH TEMPLE
SALT LAKE CITY, UTAH 84111

Docket No 13768 73

receiving, from the integrating component, an identifier that can be used by the application to access the data over the specified data transport component.

3. A method as recited in claim 2 wherein the integrating component implements a connection manager interface that may support a connection manager component and the data transport components interact with the integrating component over the connection manager interface to effectively register their respective data types so that a redirection command received over the application-level interface specifying a data type comprises the steps of interacting over the connection manager interface of the integrating component in order to determine the correct data transport component based on data type.

4. A computer-readable medium having computer-executable instructions for performing the steps recited in claim 1.

WORKMAN, NYDEGGER & SEELEY
A PROFESSIONAL CORPORATION
ATTORNEYS AT LAW
1000 EAGLE GATE TOWER
60 EAST SOUTH TEMPLE
SALT LAKE CITY, UTAH 84111

5. A connection-oriented driver subsystem where connection control information is communicated to an application through one interface while data and data control information is communicated through a transport driver, the driver subsystem comprising:

at least one simplified connection-oriented device driver controlling a connection-oriented media device;

at least one data transport protocol driver capable of communication with an application;

an integrating component that interfaces with the at least one simplified connection-oriented device driver and the at least one data transport, said at least one simplified device driver and said at least one data transport serving as clients to said integration component and said integration component:

providing an abstracted connection interface that is available to a client that allows a client to create a connection with a desired location using a media device controlled by a simplified connection-oriented device driver; and

providing facility for associating a connection created by a client through the connection interface with a data transport thereby allowing the client to send and receive data and data control information over the previously established connection; and

a proxy driver that interfaces with the connection interface and the transport interface of the integration component as a client, said proxy driver:

receiving abstract connection creation and control commands from an application and implementing such commands through appropriate use of the connection interface to create and manage a connection;

causing redirection of data and data control information from the previously created connection through the proxy driver to a designated transport designated in an abstract connection control command; and

WORKMAN, NYDEGGER & SEELEY
A PROFESSIONAL CORPORATION
ATTORNEYS AT LAW
1000 EAGLE GATE TOWER
60 EAST SOUTH TEMPLE
SALT LAKE CITY, UTAH 84111

returning to an application, in response to a previously received connection control command, an identifier to be used by the application for receiving data and data control information from the designated transport so that connection control information is communicated to the application through the proxy driver while data and data control information is communicated to the application through the designated transport.

6. A subsystem as recited in claim 5 wherein the integrating component is incorporated as part of an operating system.

WORKMAN, NYDEGGER & SEELEY
A PROFESSIONAL CORPORATION
ATTORNEYS AT LAW
1000 EAGLE GATE TOWER
60 EAST SOUTH TEMPLE
SALT LAKE CITY, UTAH 84111

7. A computer program product for interacting with a known application-level interface and an integrating component of a connection-oriented I/O subsystem in order to represent the characteristics of an underlying connection-oriented device to an application and allow an application to take advantage of the connection-oriented I/O subsystem over the known application-level interface without requiring the application programmer to program to a new interface, said computer program product comprising:

a computer-readable medium; and

computer-executable instructions carried on said computer-readable medium for performing the steps of:

representing underlying connection-oriented devices and device characteristics over a known application level interface;

receiving abstract connection creation and control commands from an application over the known application-level interface;

interacting with the integrating component to ascertain the underlying connection-oriented device and implement received connection creation and control commands.

8. A computer program product as recited in claim 7 further comprising computer-executable instructions for performing the steps of:

receiving a connection creation command in the known application-level interface format;

interacting with the integrating component to create the connection;

receiving a redirection command to send data and data control information received over the connection to a designated data transport;

causing redirection of data and data control information from the previously created connection to a designated transport; and

WORKMAN, NYDEGGER & SEELEY
A PROFESSIONAL CORPORATION
ATTORNEYS AT LAW
1000 EAGLE GATE TOWER
60 EAST SOUTH TEMPLE
SALT LAKE CITY, UTAH 84111

returning to the application an identifier to be used by the application for receiving data and data control information from the designated transport.

9. A computer program product as recited in claim 8 wherein the data redirection takes place in the integrating component.

WORKMAN, NYDEGGER & SEELEY
A PROFESSIONAL CORPORATION
ATTORNEYS AT LAW
1000 EAGLE GATE TOWER
60 EAST SOUTH TEMPLE
SALT LAKE CITY, UTAH 84111

ABSTRACT OF THE INVENTION

In an I/O subsystem comprising device drivers for controlling connection-oriented hardware adapters, connection-oriented data transports, and an integrating component allowing data transport drivers and hardware device drivers to be "bound" together to form data communication channels. The integrating component exposes a connection interface for creating and maintaining connections. Furthermore, a proxy client component is disclosed that interacts with a known application-level interface, such as TAPI, and converts such known application-level interface commands into appropriate commands for the connection interface of the integrating component. Additionally, the proxy client component will interact with the integrating component in order to represent the underlying connection-oriented devices and attendant device characteristics to the known application-level interface. All this allows the application programmer to take advantage of a connection-oriented I/O subsystem and underlying connection-oriented devices without requiring the programmer to learn yet another interface. Added benefits include the ability to dynamically redirect a data stream received over a created connection from one data transport to another under application control. This is useful in multimedia applications where different types of data may come over the same connection.

G:\DATA\PAT\1376873 PAT

WORKMAN, NYDEGGER & SEELEY
A PROFESSIONAL CORPORATION
ATTORNEYS AT LAW
1000 EAGLE GATE TOWER
60 EAST SOUTH TEMPLE
SALT LAKE CITY, UTAH 84111

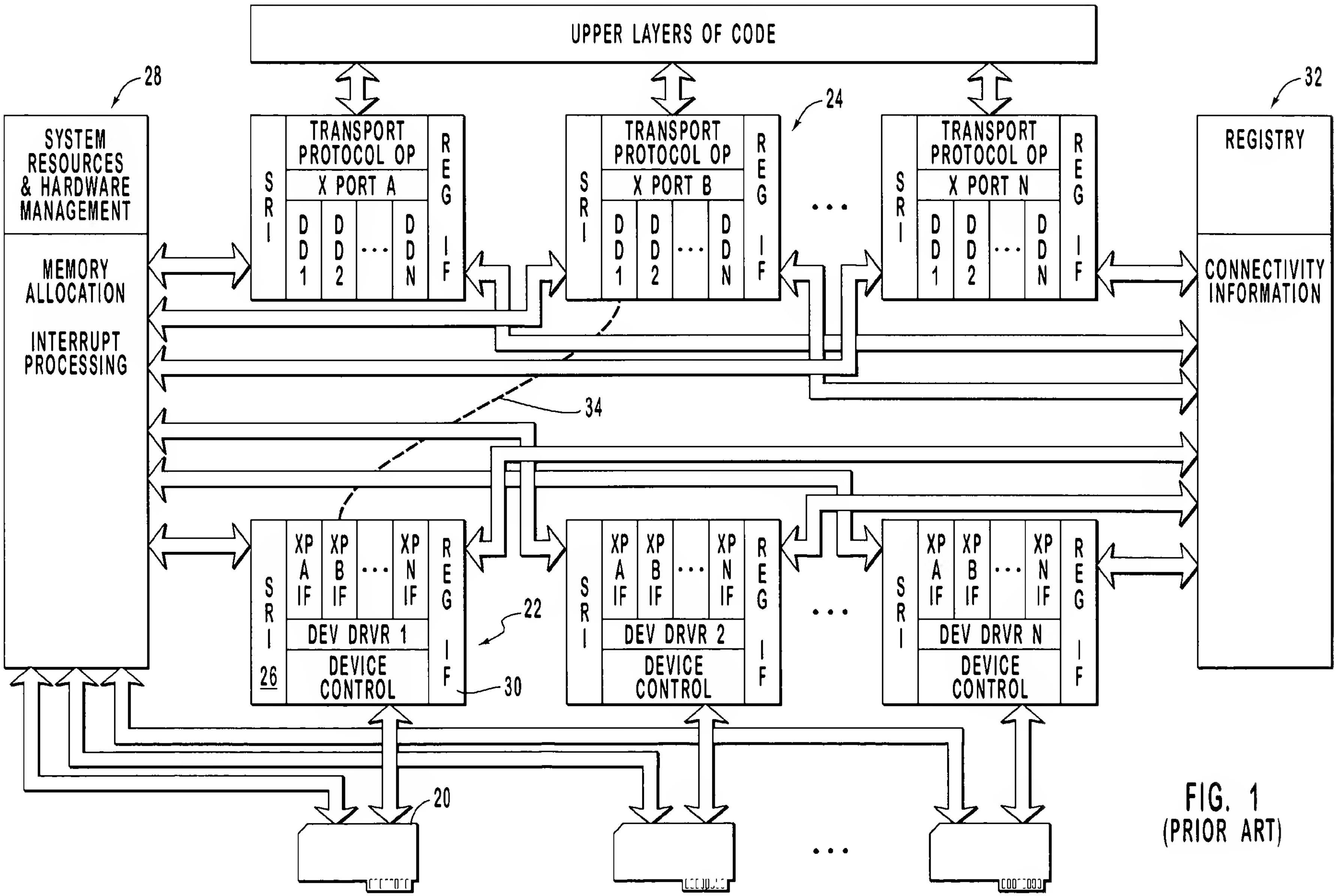


FIG. 1
(PRIOR ART)

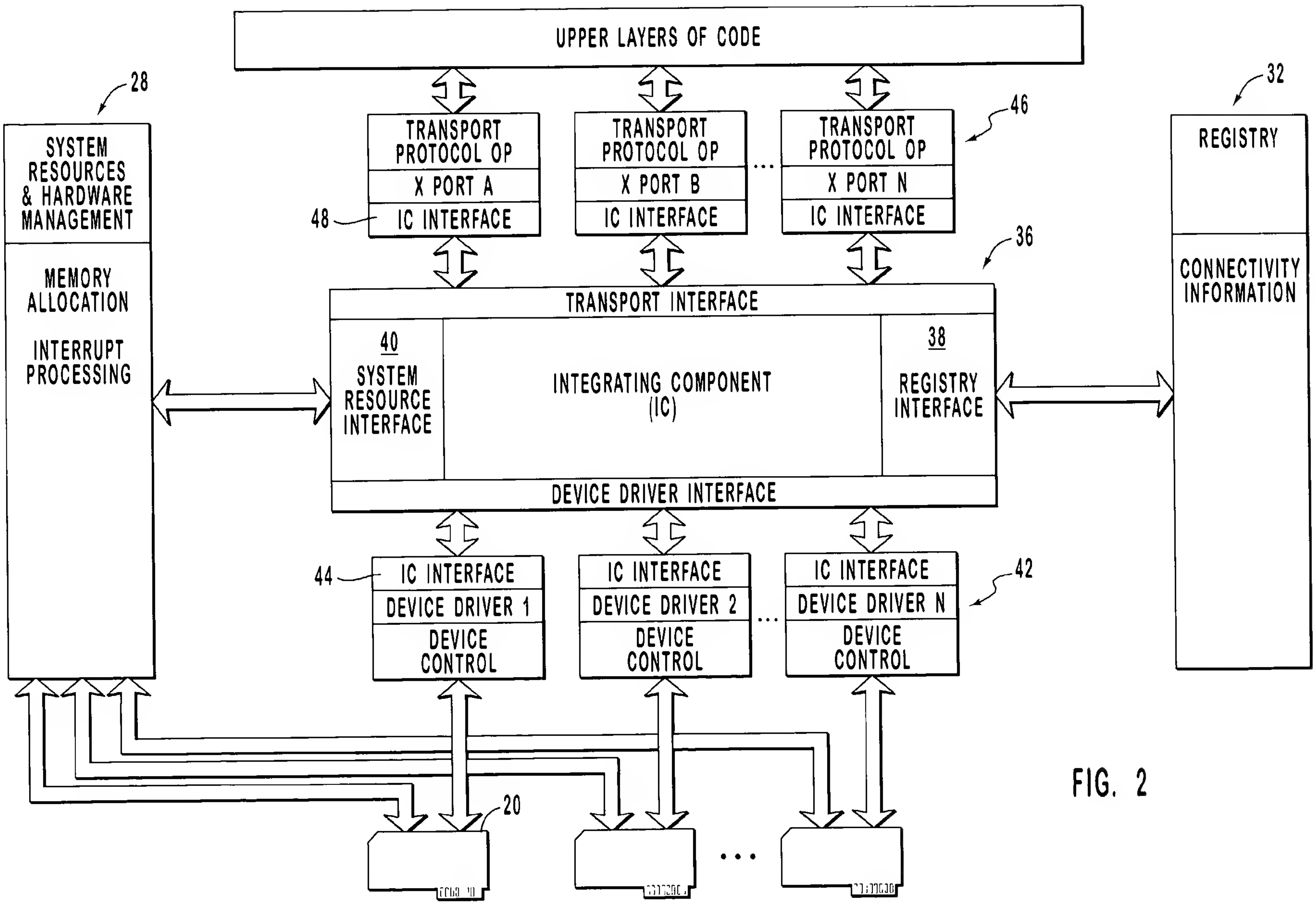
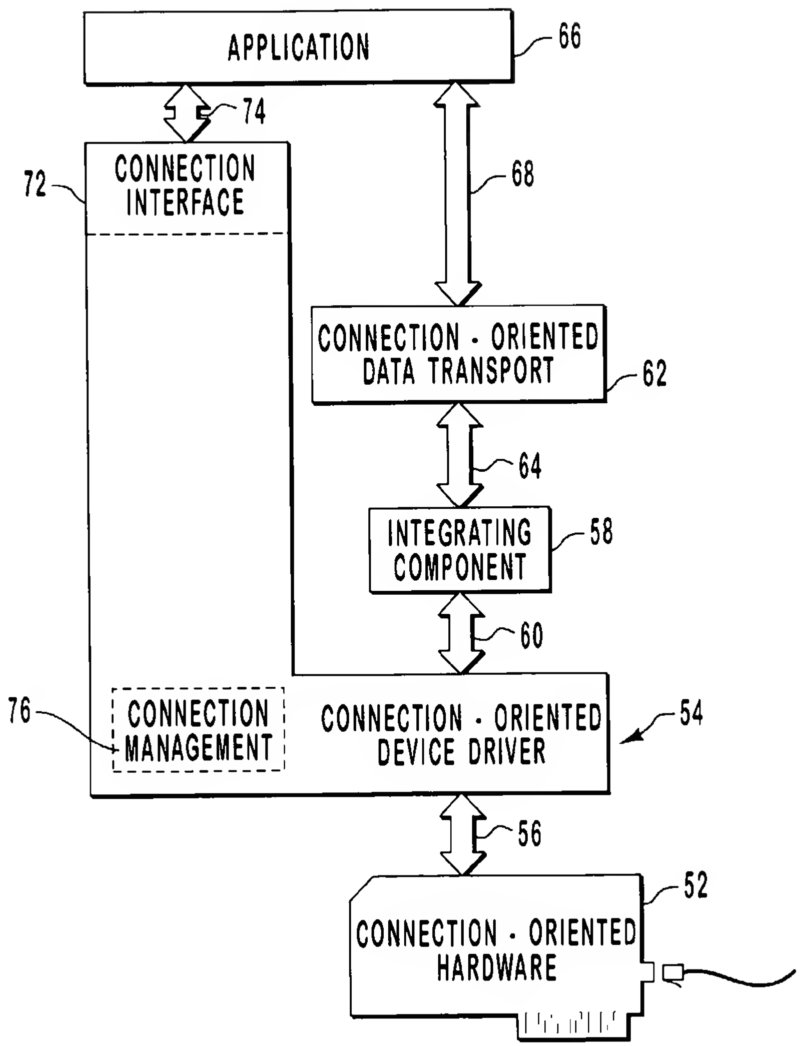
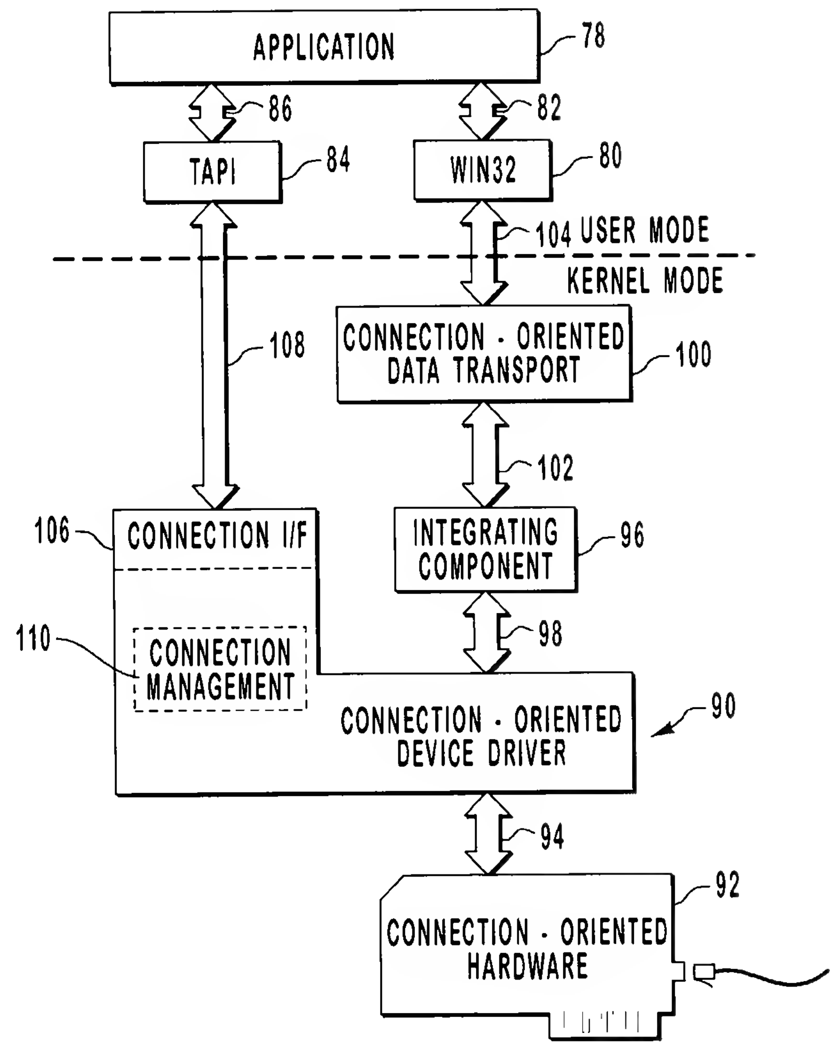

UPPER LAYERS OF CODE
28
SYSTEM RESOURCES & HARDWARE MANAGEMENT
MEMORY ALLOCATION
INTERRUPT PROCESSING
TRANSPORT PROTOCOL OP
X PORT A
IC INTERFACE
48
TRANSPORT PROTOCOL OP
X PORT B
IC INTERFACE
TRANSPORT PROTOCOL OP
X PORT N
IC INTERFACE
46
36
TRANSPORT INTERFACE
40
SYSTEM RESOURCE INTERFACE
INTEGRATING COMPONENT (IC)
38
REGISTRY INTERFACE
DEVICE DRIVER INTERFACE
44
IC INTERFACE
DEVICE DRIVER 1
DEVICE CONTROL
IC INTERFACE
DEVICE DRIVER 2
DEVICE CONTROL
IC INTERFACE
DEVICE DRIVER N
DEVICE CONTROL
42
20
32
REGISTRY
CONNECTIVITY INFORMATION

FIG. 2



APPLICATION
66
74
68
CONNECTION INTERFACE
72
CONNECTION - ORIENTED DATA TRANSPORT
62
64
INTEGRATING COMPONENT
58
60
76
CONNECTION MANAGEMENT
CONNECTION - ORIENTED DEVICE DRIVER
54
56
52
CONNECTION - ORIENTED HARDWARE

FIG. 3



APPLICATION
78
86
82
TAPI
84
WIN32
80
104
USER MODE
KERNEL MODE
108
CONNECTION - ORIENTED DATA TRANSPORT
100
102
106
CONNECTION I/F
INTEGRATING COMPONENT
96
110
CONNECTION MANAGEMENT
98
CONNECTION - ORIENTED DEVICE DRIVER
90
94
92
CONNECTION - ORIENTED HARDWARE

FIG. 4

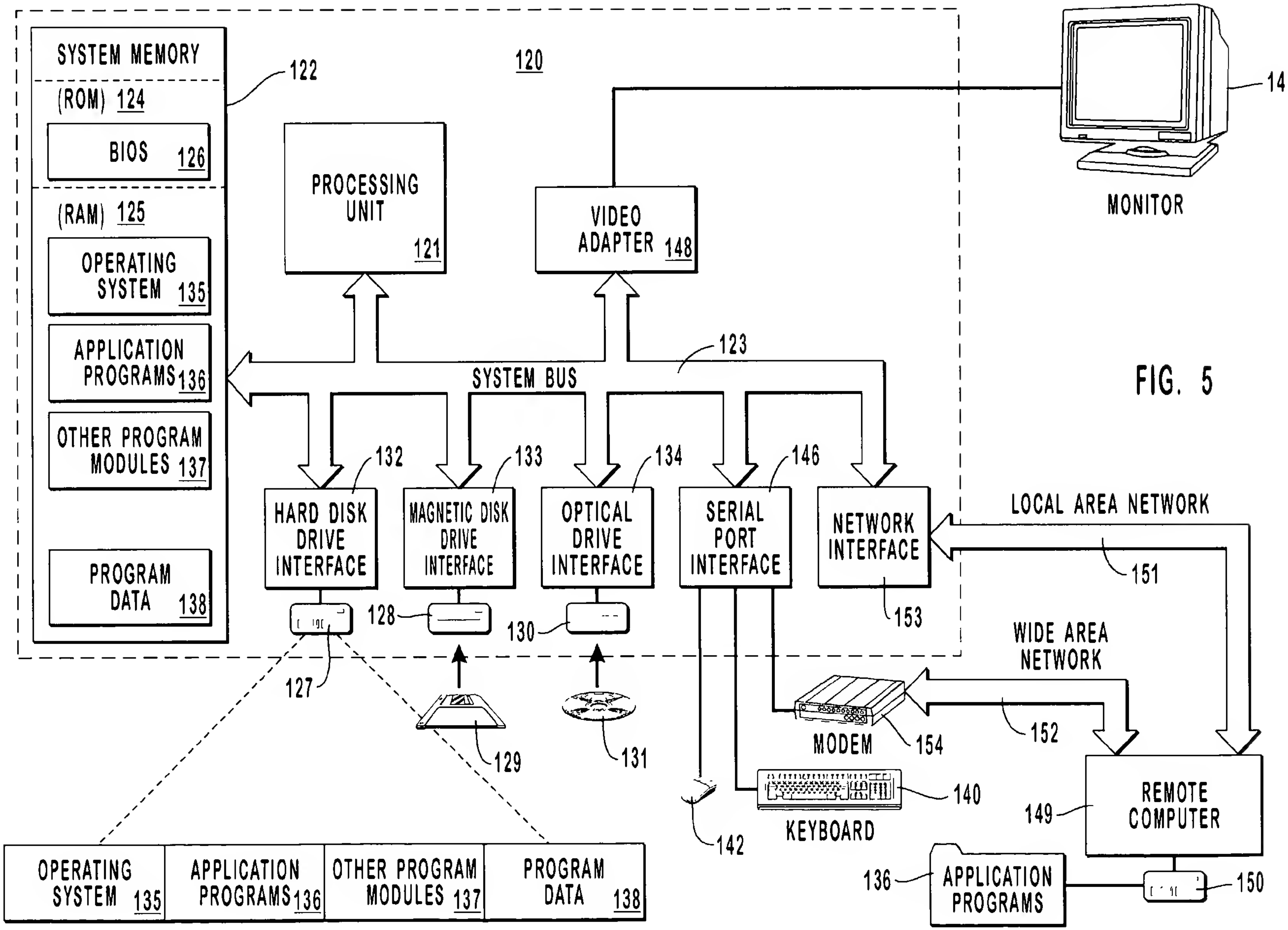

SYSTEM MEMORY
(ROM) 124
BIOS 126
(RAM) 125
OPERATING SYSTEM 135
APPLICATION PROGRAMS 136
OTHER PROGRAM MODULES 137
PROGRAM DATA 138
122
120
PROCESSING UNIT 121
VIDEO ADAPTER 148
MONITOR
147
SYSTEM BUS
123
HARD DISK DRIVE INTERFACE
132
MAGNETIC DISK DRIVE INTERFACE
133
OPTICAL DRIVE INTERFACE
134
SERIAL PORT INTERFACE
146
NETWORK INTERFACE
153
LOCAL AREA NETWORK
151
WIDE AREA NETWORK
152
127
128
129
130
131
142
MODEM
154
KEYBOARD
140
REMOTE COMPUTER
149
APPLICATION PROGRAMS
136
150
OPERATING SYSTEM 135
APPLICATION PROGRAMS 136
OTHER PROGRAM MODULES 137
PROGRAM DATA 138

FIG. 5

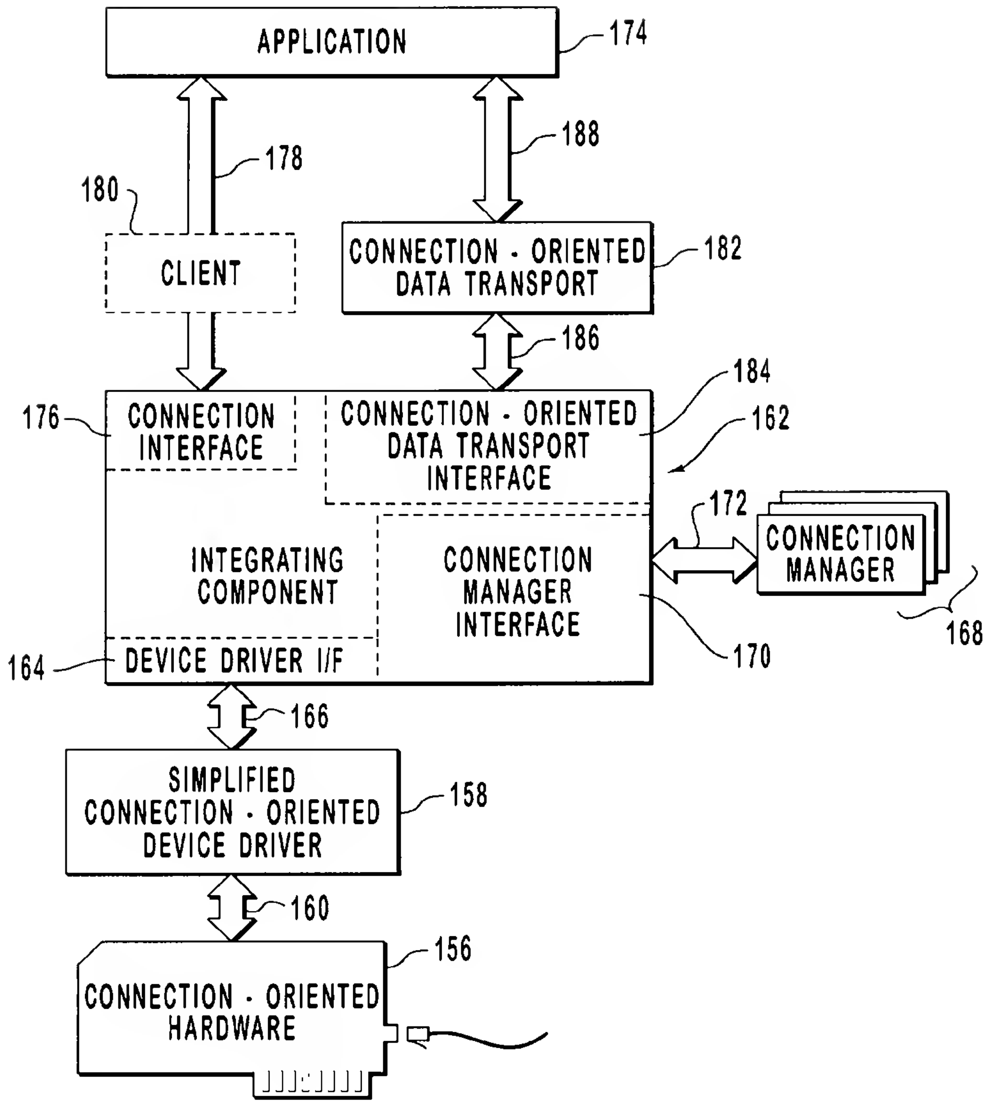

APPLICATION
174
178
188
180
CLIENT
CONNECTION - ORIENTED DATA TRANSPORT
182
186
184
176
CONNECTION INTERFACE
CONNECTION - ORIENTED DATA TRANSPORT INTERFACE
162
172
CONNECTION MANAGER
INTEGRATING COMPONENT
CONNECTION MANAGER INTERFACE
168
164
DEVICE DRIVER I/F
170
166
SIMPLIFIED CONNECTION - ORIENTED DEVICE DRIVER
158
160
156
CONNECTION - ORIENTED HARDWARE

FIG. 6

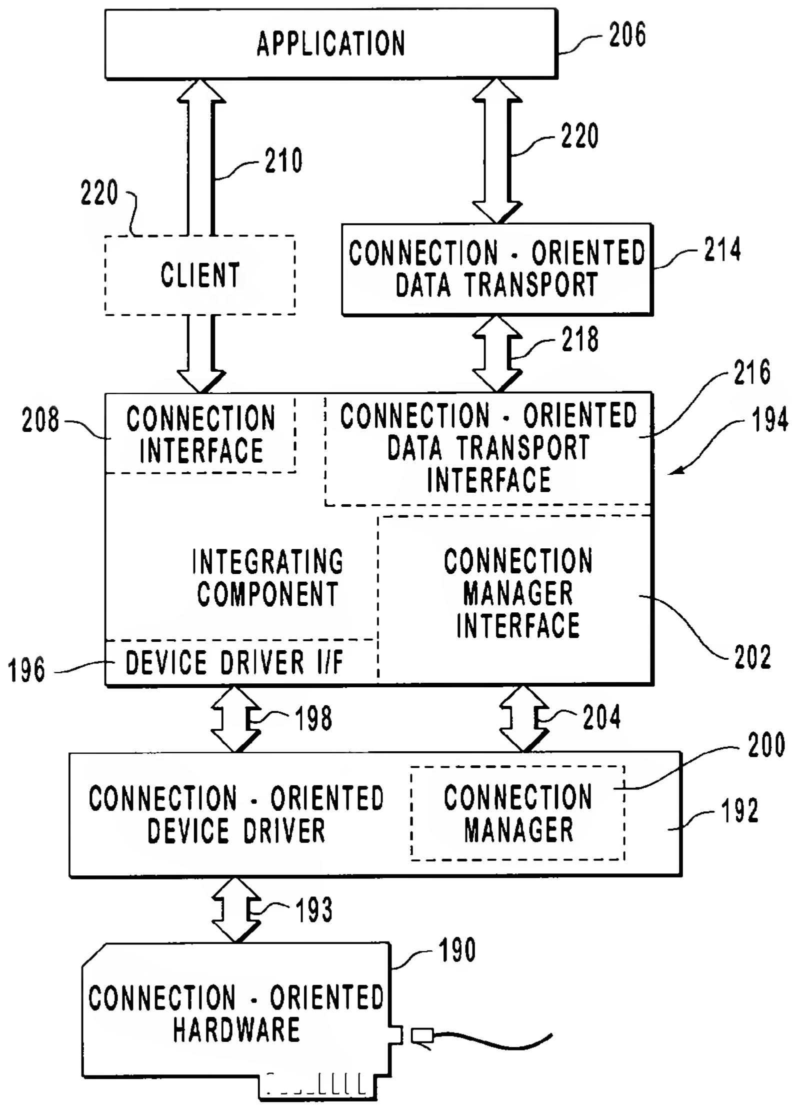

APPLICATION
206
220
210
220
CLIENT
CONNECTION - ORIENTED DATA TRANSPORT
214
218
216
194
208
CONNECTION INTERFACE
CONNECTION - ORIENTED DATA TRANSPORT INTERFACE
INTEGRATING COMPONENT
CONNECTION MANAGER INTERFACE
196
DEVICE DRIVER I/F
202
198
204
200
CONNECTION - ORIENTED DEVICE DRIVER
CONNECTION MANAGER
192
193
190
CONNECTION - ORIENTED HARDWARE

FIG. 7

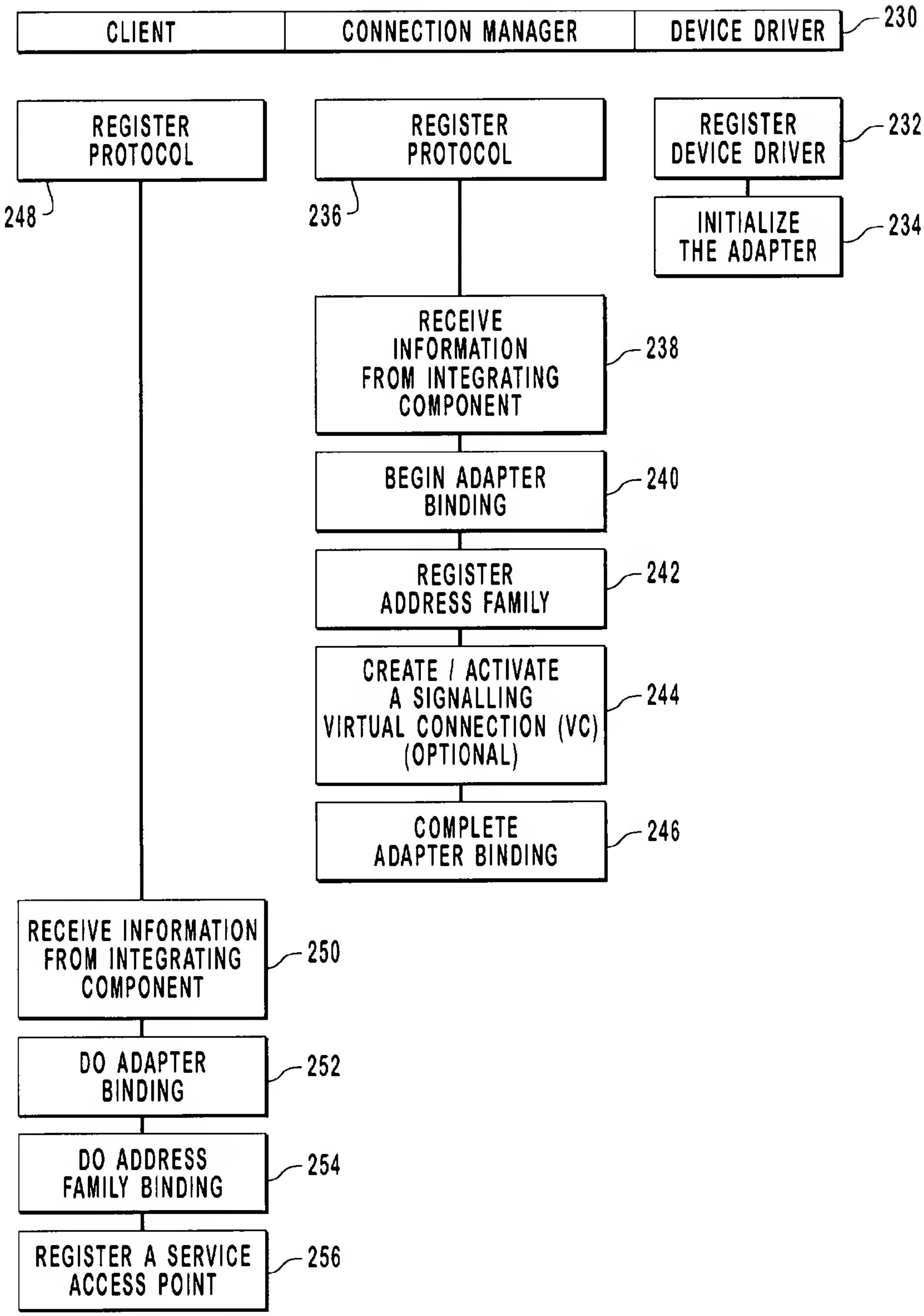

CLIENT
CONNECTION MANAGER
DEVICE DRIVER
230
REGISTER PROTOCOL
248
REGISTER PROTOCOL
236
REGISTER DEVICE DRIVER
232
INITIALIZE THE ADAPTER
234
RECEIVE INFORMATION FROM INTEGRATING COMPONENT
238
BEGIN ADAPTER BINDING
240
REGISTER ADDRESS FAMILY
242
CREATE / ACTIVATE A SIGNALLING VIRTUAL CONNECTION (VC) (OPTIONAL)
244
COMPLETE ADAPTER BINDING
246
RECEIVE INFORMATION FROM INTEGRATING COMPONENT
250
DO ADAPTER BINDING
252
DO ADDRESS FAMILY BINDING
254
REGISTER A SERVICE ACCESS POINT
256

FIG. 8

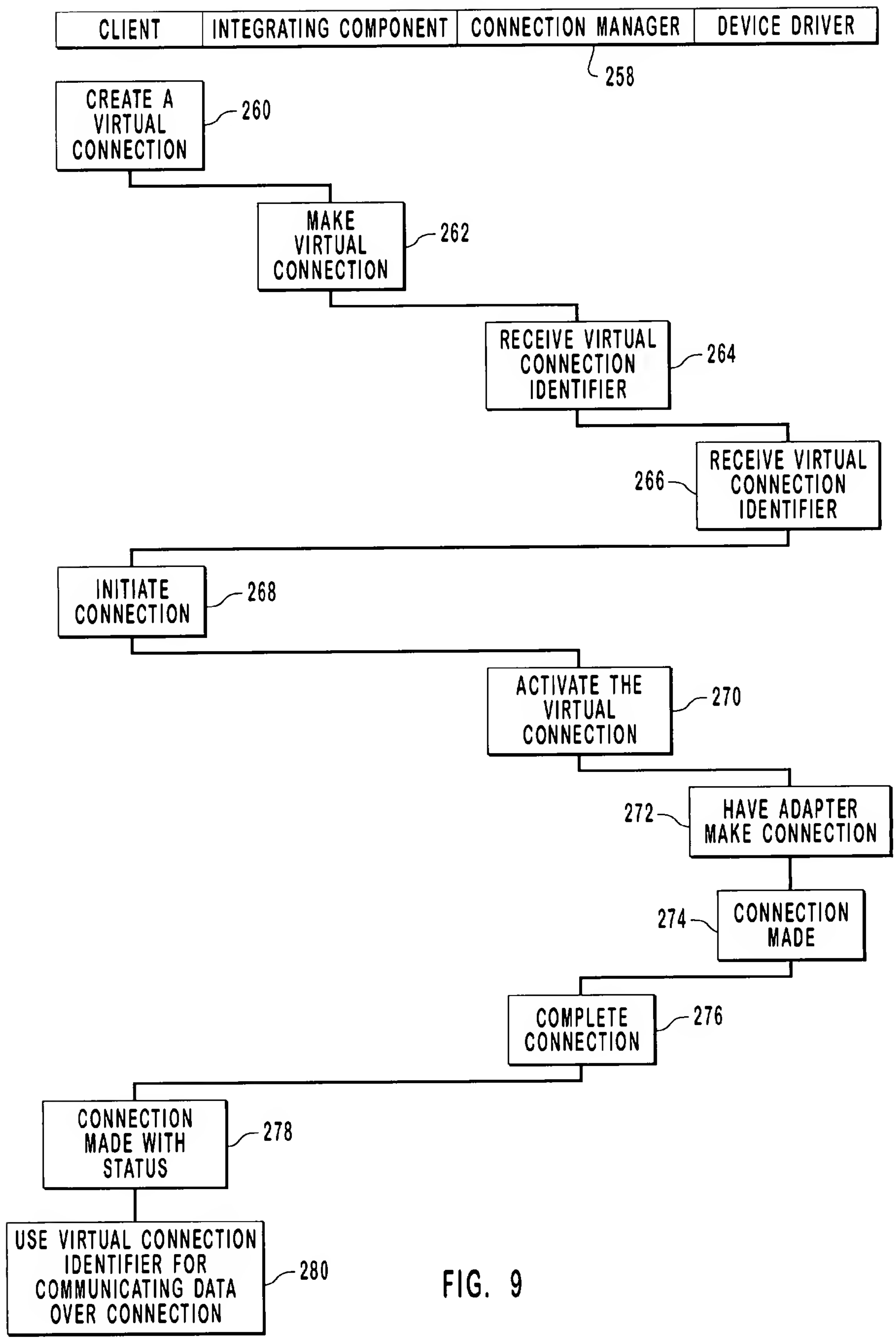
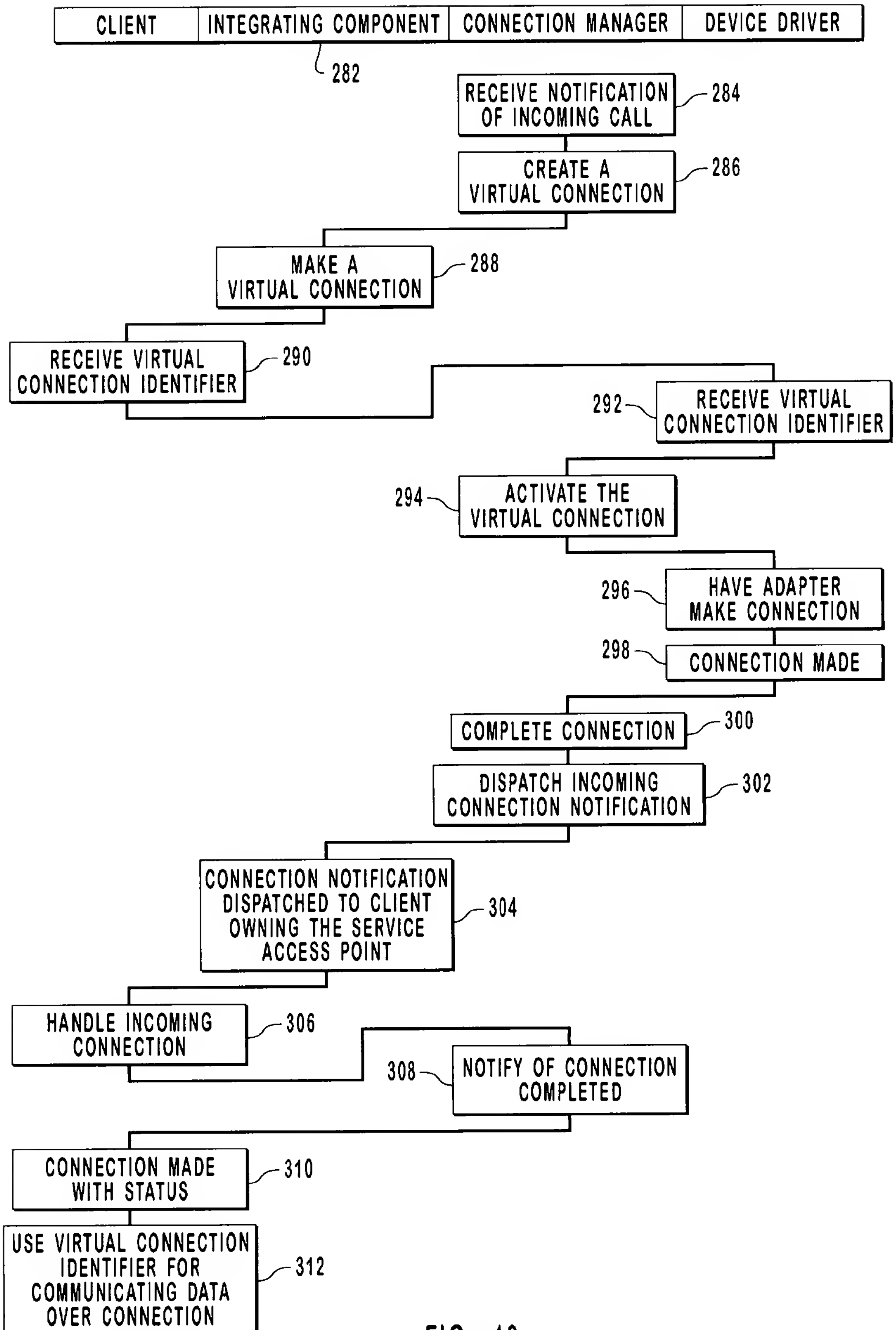

CLIENT
INTEGRATING COMPONENT
CONNECTION MANAGER
DEVICE DRIVER
258
CREATE A VIRTUAL CONNECTION
260
MAKE VIRTUAL CONNECTION
262
RECEIVE VIRTUAL CONNECTION IDENTIFIER
264
RECEIVE VIRTUAL CONNECTION IDENTIFIER
266
INITIATE CONNECTION
268
ACTIVATE THE VIRTUAL CONNECTION
270
HAVE ADAPTER MAKE CONNECTION
272
CONNECTION MADE
274
COMPLETE CONNECTION
276
CONNECTION MADE WITH STATUS
278
USE VIRTUAL CONNECTION IDENTIFIER FOR COMMUNICATING DATA OVER CONNECTION
280

FIG. 9



CLIENT
INTEGRATING COMPONENT
CONNECTION MANAGER
DEVICE DRIVER
282
RECEIVE NOTIFICATION OF INCOMING CALL
284
CREATE A VIRTUAL CONNECTION
286
MAKE A VIRTUAL CONNECTION
288
RECEIVE VIRTUAL CONNECTION IDENTIFIER
290
292
RECEIVE VIRTUAL CONNECTION IDENTIFIER
294
ACTIVATE THE VIRTUAL CONNECTION
296
HAVE ADAPTER MAKE CONNECTION
298
CONNECTION MADE
COMPLETE CONNECTION
300
DISPATCH INCOMING CONNECTION NOTIFICATION
302
CONNECTION NOTIFICATION DISPATCHED TO CLIENT OWNING THE SERVICE ACCESS POINT
304
HANDLE INCOMING CONNECTION
306
308
NOTIFY OF CONNECTION COMPLETED
CONNECTION MADE WITH STATUS
310
USE VIRTUAL CONNECTION IDENTIFIER FOR COMMUNICATING DATA OVER CONNECTION
312

FIG. 10

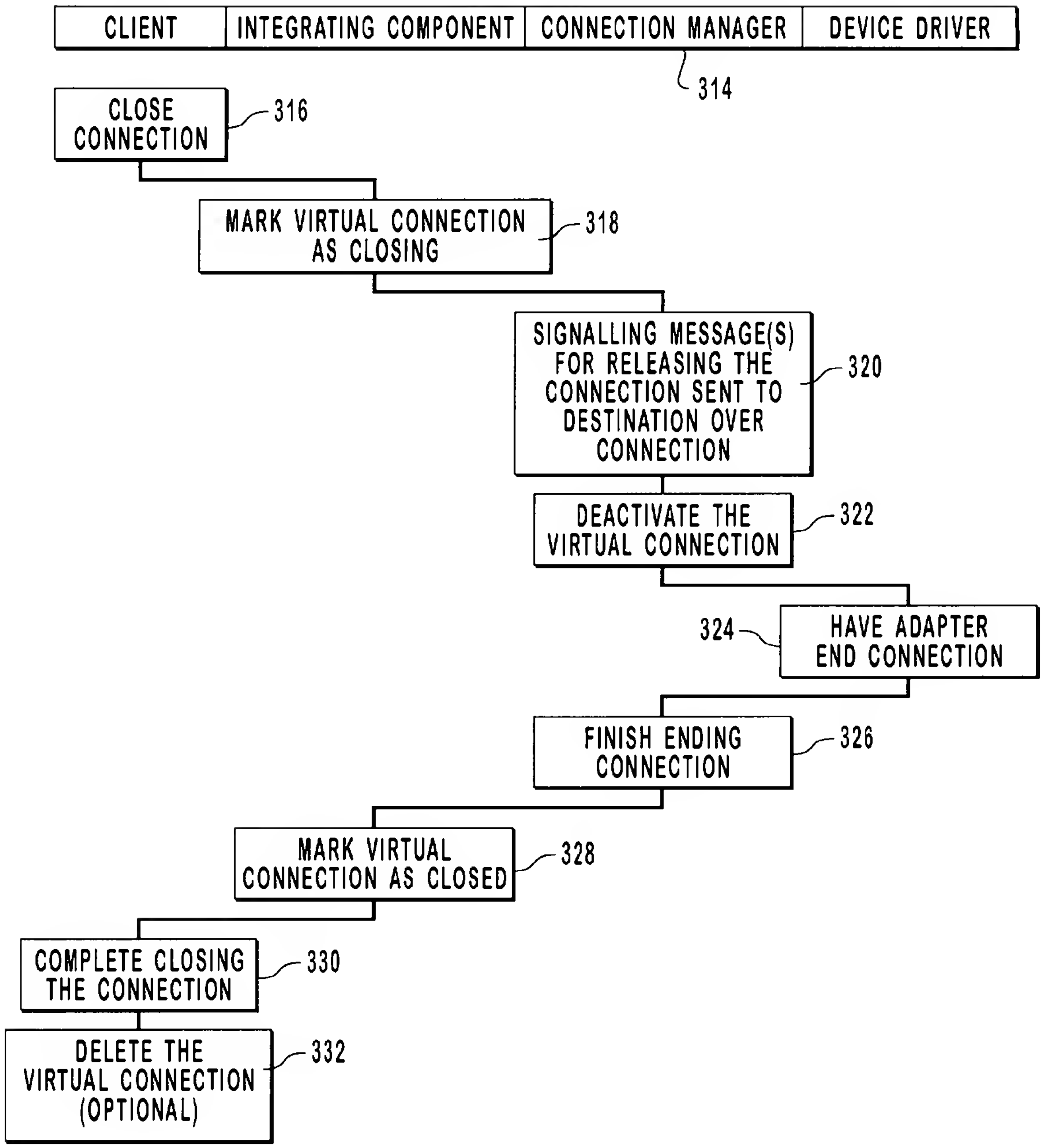

CLIENT
INTEGRATING COMPONENT
CONNECTION MANAGER
DEVICE DRIVER
314
CLOSE CONNECTION
316
MARK VIRTUAL CONNECTION AS CLOSING
318
SIGNALLING MESSAGE(S) FOR RELEASING THE CONNECTION SENT TO DESTINATION OVER CONNECTION
320
DEACTIVATE THE VIRTUAL CONNECTION
322
324
HAVE ADAPTER END CONNECTION
FINISH ENDING CONNECTION
326
MARK VIRTUAL CONNECTION AS CLOSED
328
COMPLETE CLOSING THE CONNECTION
330
DELETE THE VIRTUAL CONNECTION (OPTIONAL)
332

FIG. 11

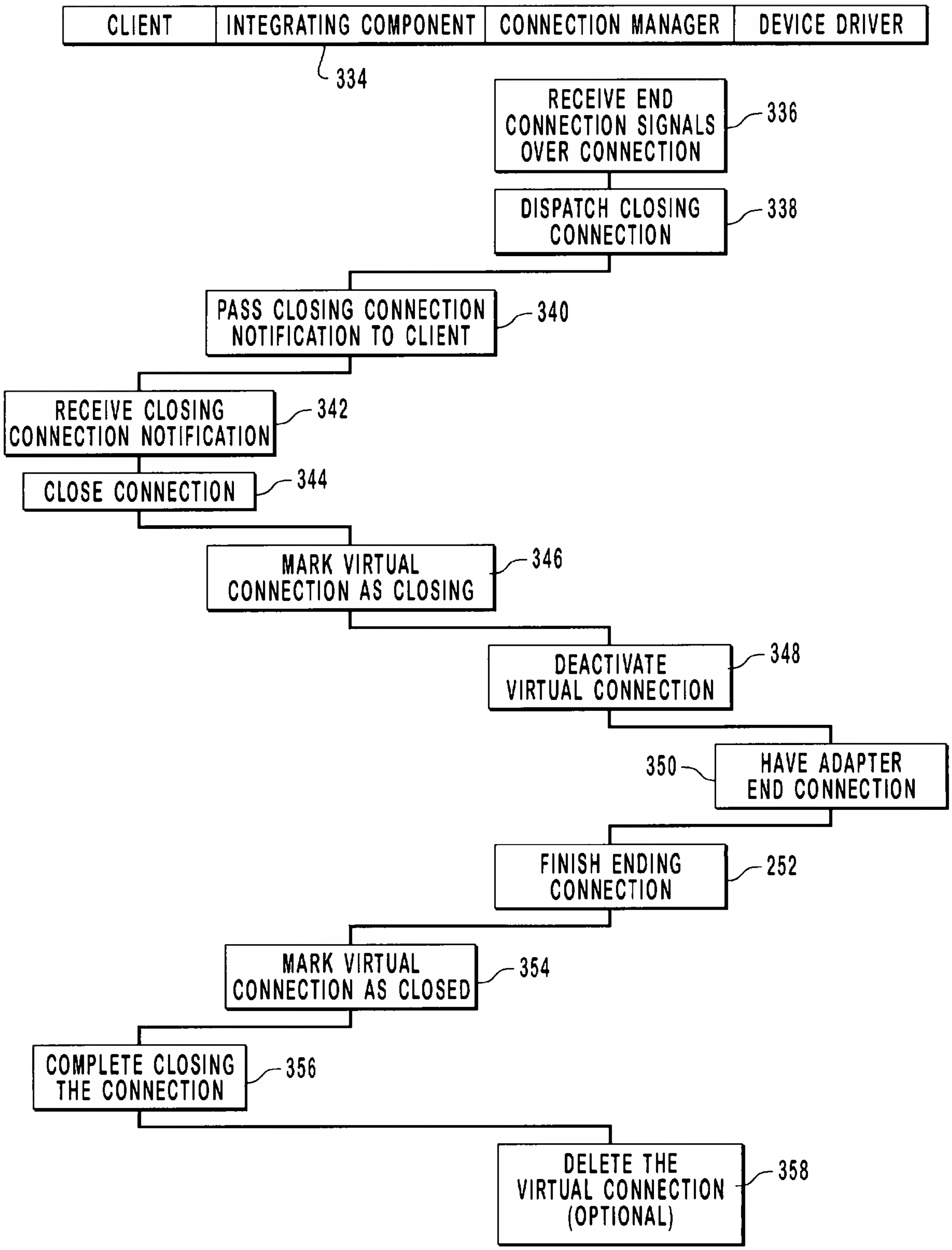

CLIENT
INTEGRATING COMPONENT
CONNECTION MANAGER
DEVICE DRIVER
334
RECEIVE END CONNECTION SIGNALS OVER CONNECTION
336
DISPATCH CLOSING CONNECTION
338
PASS CLOSING CONNECTION NOTIFICATION TO CLIENT
340
RECEIVE CLOSING CONNECTION NOTIFICATION
342
CLOSE CONNECTION
344
MARK VIRTUAL CONNECTION AS CLOSING
346
DEACTIVATE VIRTUAL CONNECTION
348
350
HAVE ADAPTER END CONNECTION
FINISH ENDING CONNECTION
252
MARK VIRTUAL CONNECTION AS CLOSED
354
COMPLETE CLOSING THE CONNECTION
356
DELETE THE VIRTUAL CONNECTION (OPTIONAL)
358

FIG. 12

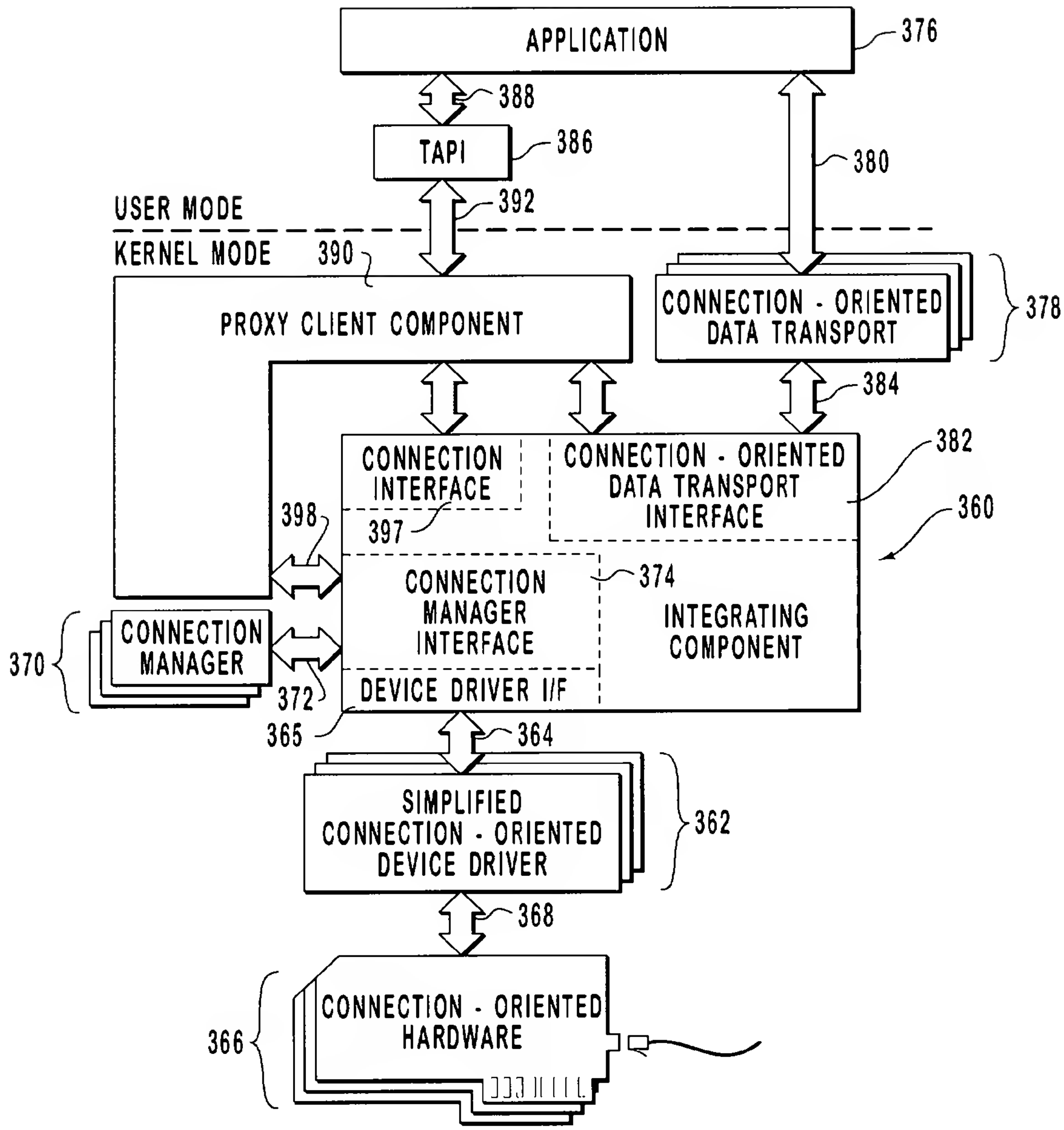

APPLICATION
376
388
TAPI
386
380
USER MODE
392
KERNEL MODE
390
PROXY CLIENT COMPONENT
CONNECTION - ORIENTED DATA TRANSPORT
378
384
CONNECTION INTERFACE
CONNECTION - ORIENTED DATA TRANSPORT INTERFACE
382
360
398
397
CONNECTION MANAGER INTERFACE
374
INTEGRATING COMPONENT
370
CONNECTION MANAGER
372
DEVICE DRIVER I/F
365
364
SIMPLIFIED CONNECTION - ORIENTED DEVICE DRIVER
362
368
CONNECTION - ORIENTED HARDWARE
366

FIG. 13

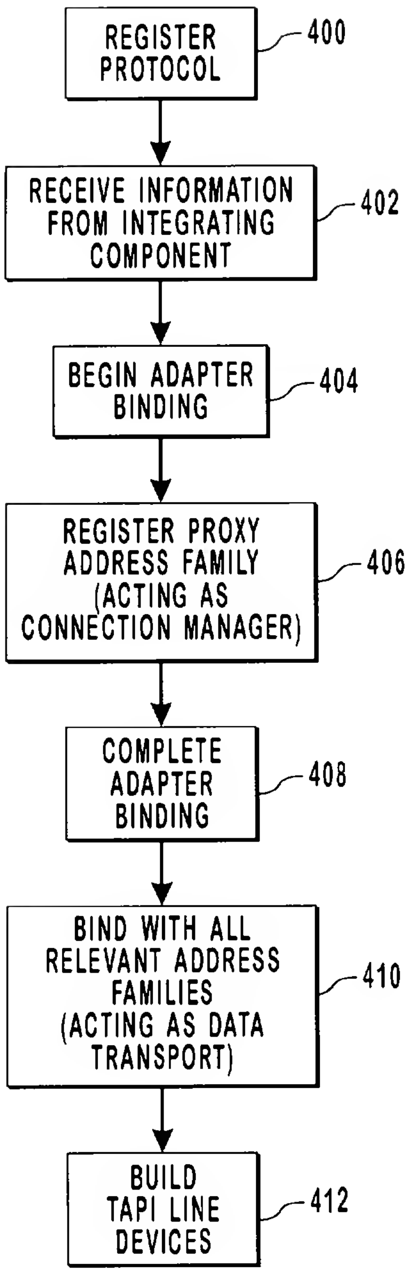

REGISTER PROTOCOL
400
RECEIVE INFORMATION FROM INTEGRATING COMPONENT
402
BEGIN ADAPTER BINDING
404
REGISTER PROXY ADDRESS FAMILY (ACTING AS CONNECTION MANAGER)
406
COMPLETE ADAPTER BINDING
408
BIND WITH ALL RELEVANT ADDRESS FAMILIES (ACTING AS DATA TRANSPORT)
410
BUILD TAPI LINE DEVICES
412

FIG. 14

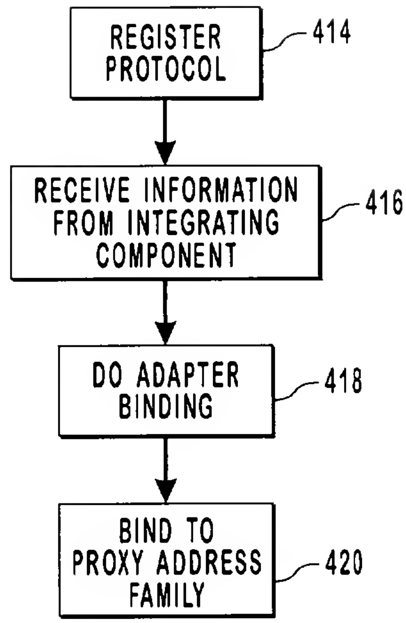

REGISTER PROTOCOL
414
RECEIVE INFORMATION FROM INTEGRATING COMPONENT
416
DO ADAPTER BINDING
418
BIND TO PROXY ADDRESS FAMILY
420

FIG. 15

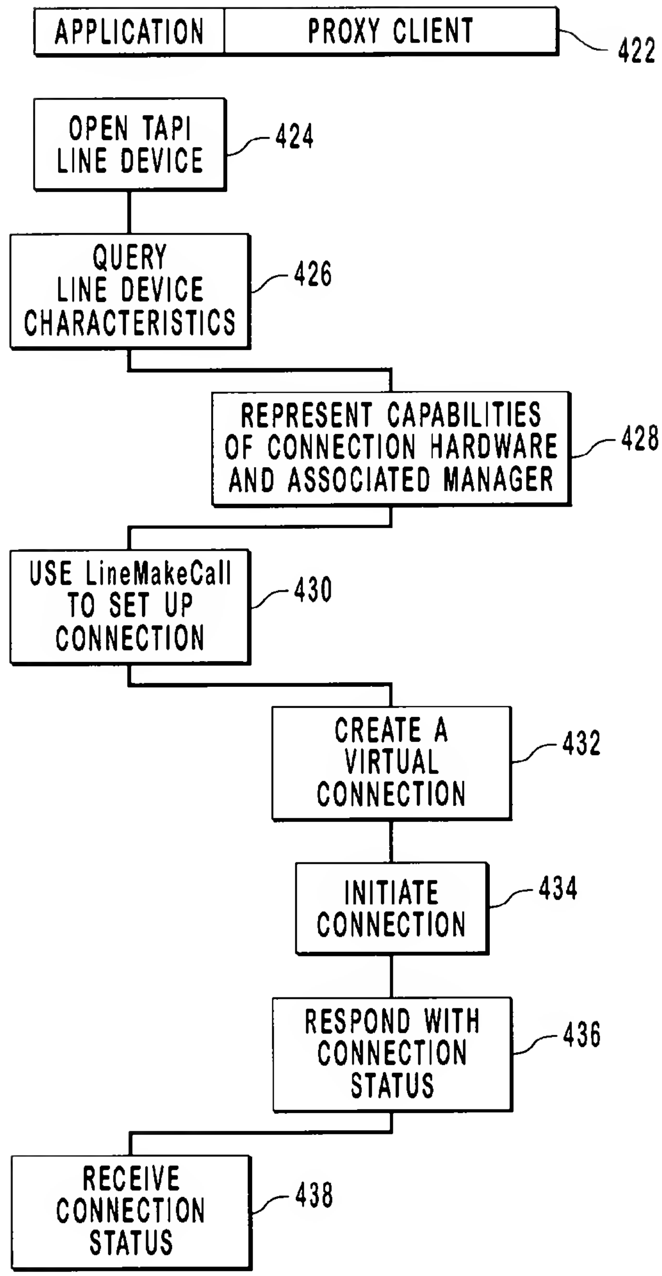

APPLICATION
PROXY CLIENT
422
OPEN TAPI LINE DEVICE
424
QUERY LINE DEVICE CHARACTERISTICS
426
REPRESENT CAPABILITIES OF CONNECTION HARDWARE AND ASSOCIATED MANAGER
428
USE LineMakeCall TO SET UP CONNECTION
430
CREATE A VIRTUAL CONNECTION
432
INITIATE CONNECTION
434
RESPOND WITH CONNECTION STATUS
436
RECEIVE CONNECTION STATUS
438

FIG. 16

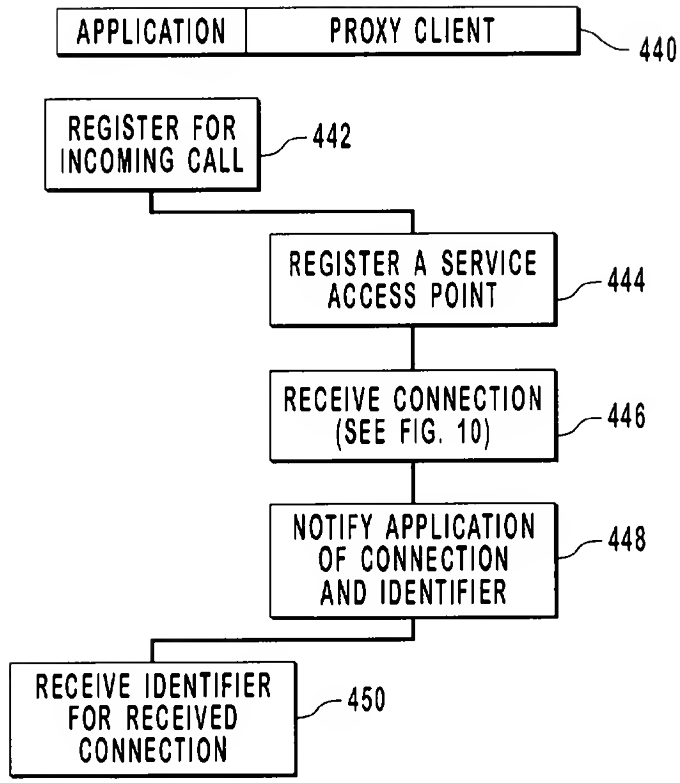

APPLICATION
PROXY CLIENT
440
REGISTER FOR INCOMING CALL
442
REGISTER A SERVICE ACCESS POINT
444
RECEIVE CONNECTION (SEE FIG. 10)
446
NOTIFY APPLICATION OF CONNECTION AND IDENTIFIER
448
RECEIVE IDENTIFIER FOR RECEIVED CONNECTION
450

FIG. 17

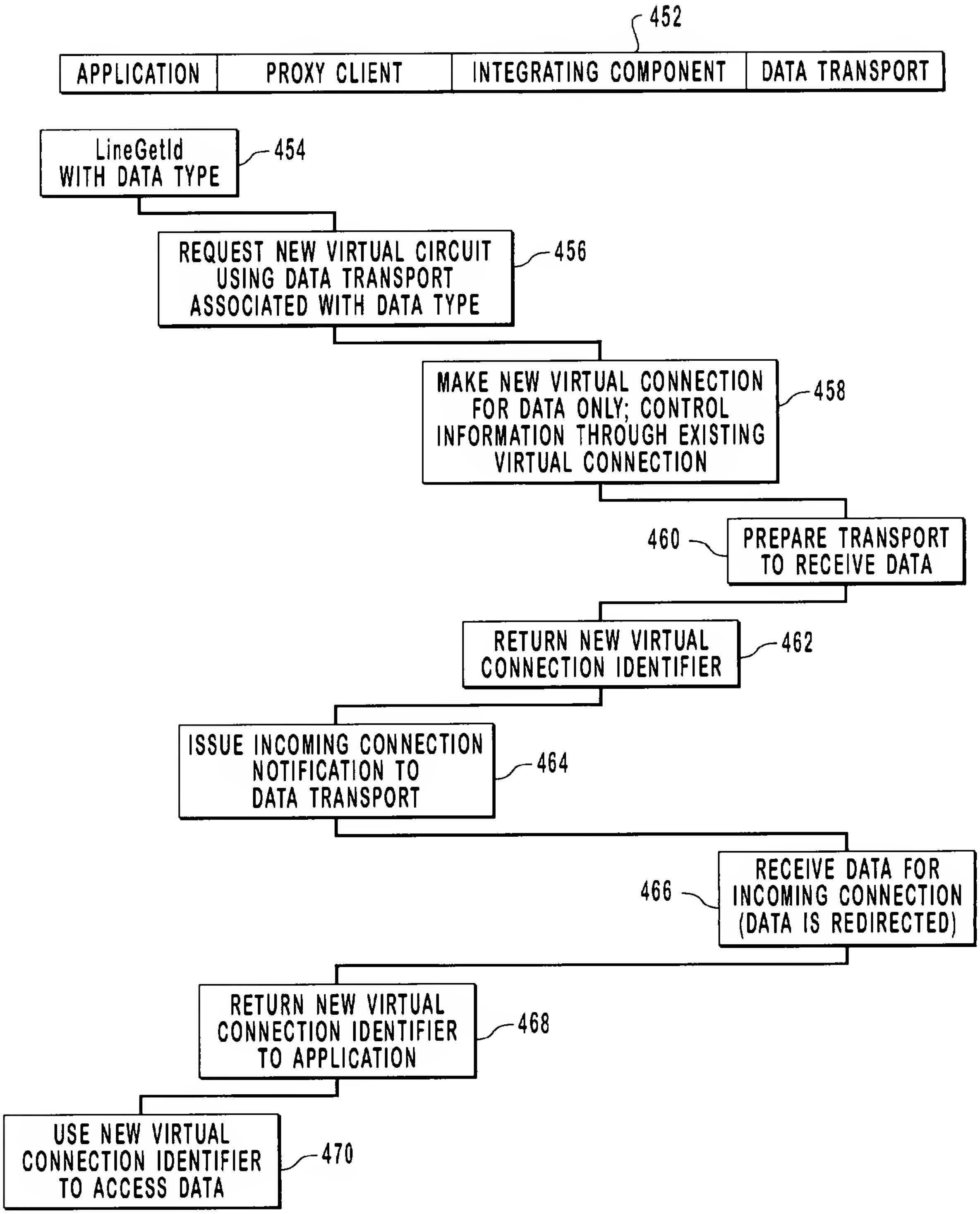

452
APPLICATION
PROXY CLIENT
INTEGRATING COMPONENT
DATA TRANSPORT
LineGetId WITH DATA TYPE
454
REQUEST NEW VIRTUAL CIRCUIT USING DATA TRANSPORT ASSOCIATED WITH DATA TYPE
456
MAKE NEW VIRTUAL CONNECTION FOR DATA ONLY; CONTROL INFORMATION THROUGH EXISTING VIRTUAL CONNECTION
458
460
PREPARE TRANSPORT TO RECEIVE DATA
RETURN NEW VIRTUAL CONNECTION IDENTIFIER
462
ISSUE INCOMING CONNECTION NOTIFICATION TO DATA TRANSPORT
464
466
RECEIVE DATA FOR INCOMING CONNECTION (DATA IS REDIRECTED)
RETURN NEW VIRTUAL CONNECTION IDENTIFIER TO APPLICATION
468
USE NEW VIRTUAL CONNECTION IDENTIFIER TO ACCESS DATA
470

FIG. 18

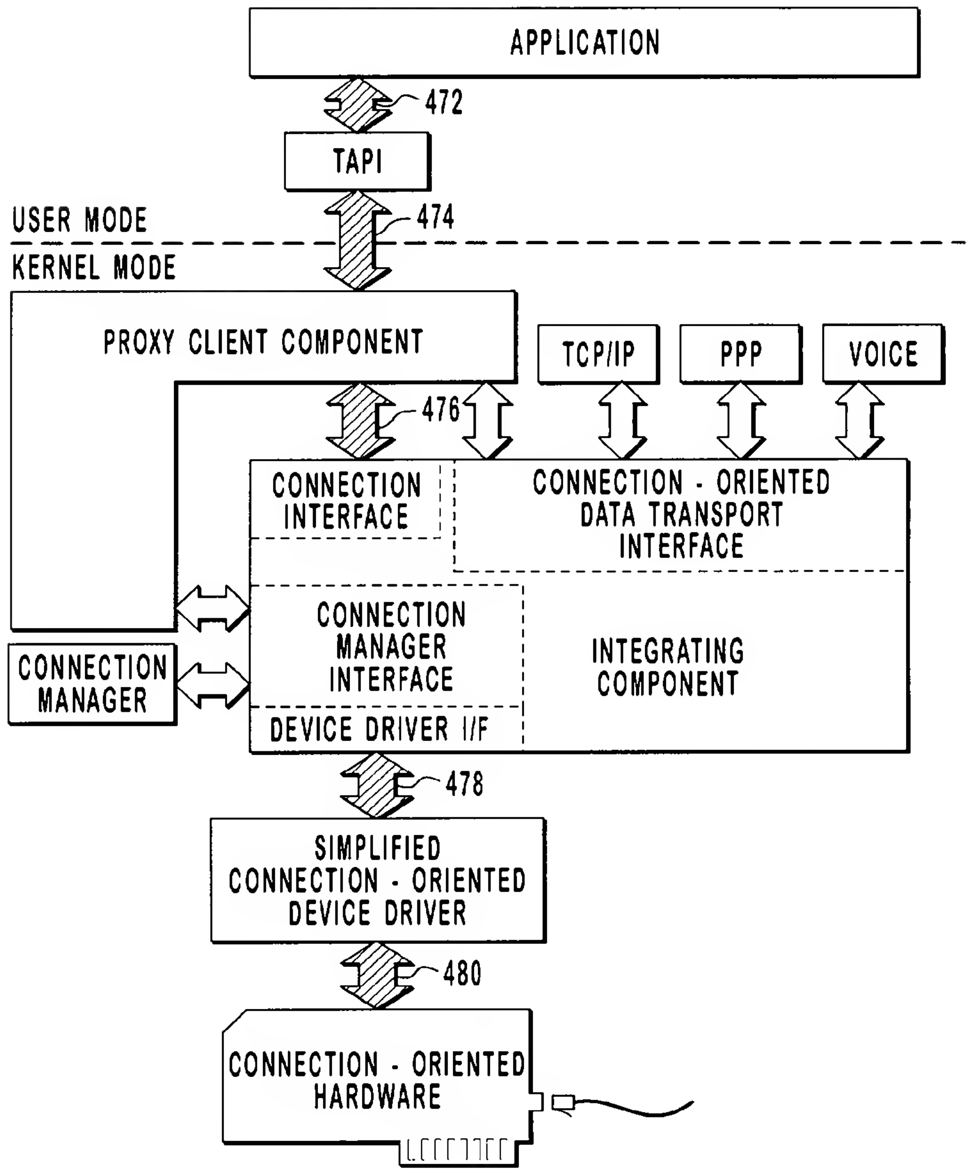

APPLICATION
472
TAPI
474
USER MODE
KERNEL MODE
PROXY CLIENT COMPONENT
TCP/IP
PPP
VOICE
476
CONNECTION INTERFACE
CONNECTION - ORIENTED DATA TRANSPORT INTERFACE
CONNECTION MANAGER INTERFACE
INTEGRATING COMPONENT
CONNECTION MANAGER
DEVICE DRIVER I/F
478
SIMPLIFIED CONNECTION - ORIENTED DEVICE DRIVER
480
CONNECTION - ORIENTED HARDWARE

FIG. 19A

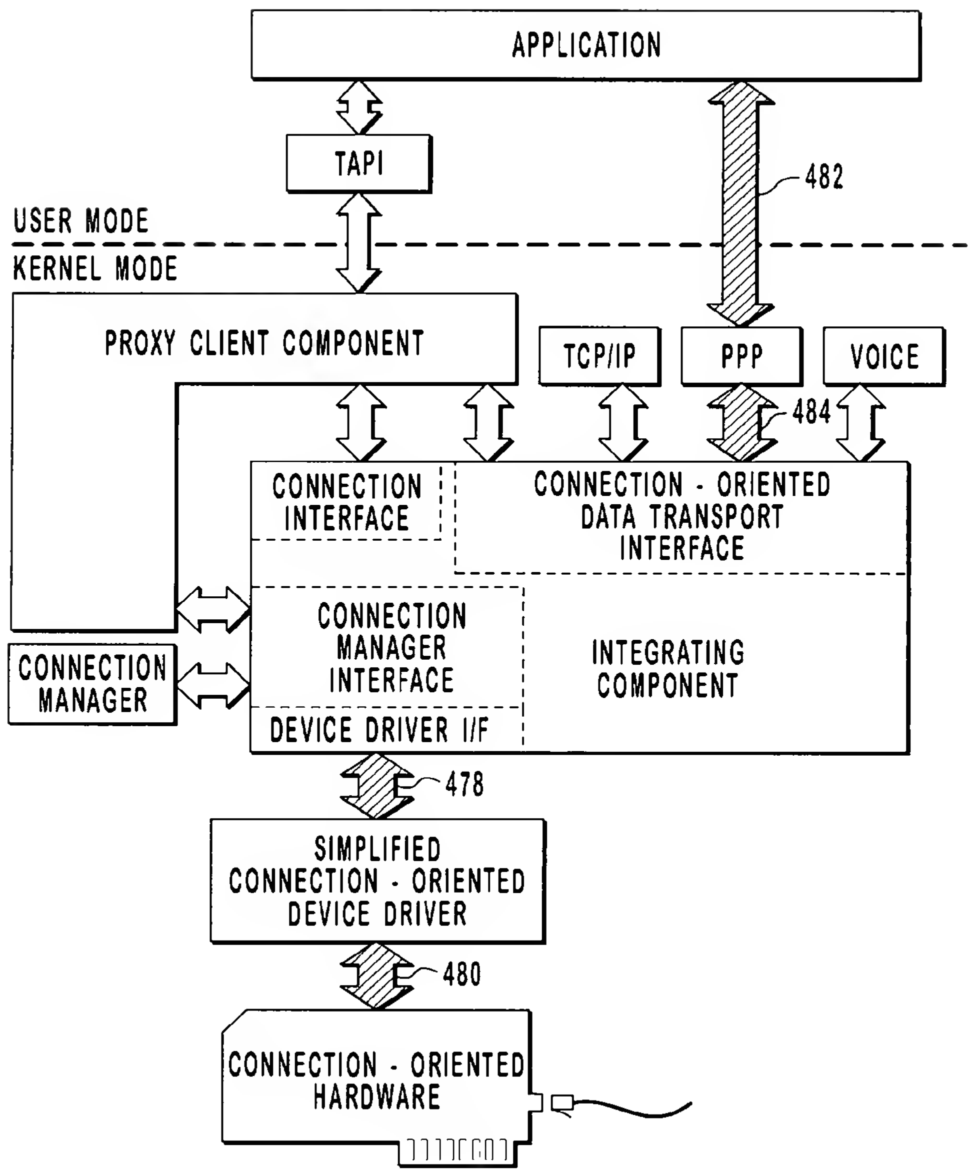

APPLICATION
TAPI
482
USER MODE
KERNEL MODE
PROXY CLIENT COMPONENT
TCP/IP
PPP
VOICE
484
CONNECTION INTERFACE
CONNECTION - ORIENTED DATA TRANSPORT INTERFACE
CONNECTION MANAGER
CONNECTION MANAGER INTERFACE
INTEGRATING COMPONENT
DEVICE DRIVER I/F
478
SIMPLIFIED CONNECTION - ORIENTED DEVICE DRIVER
480
CONNECTION - ORIENTED HARDWARE

FIG. 19B

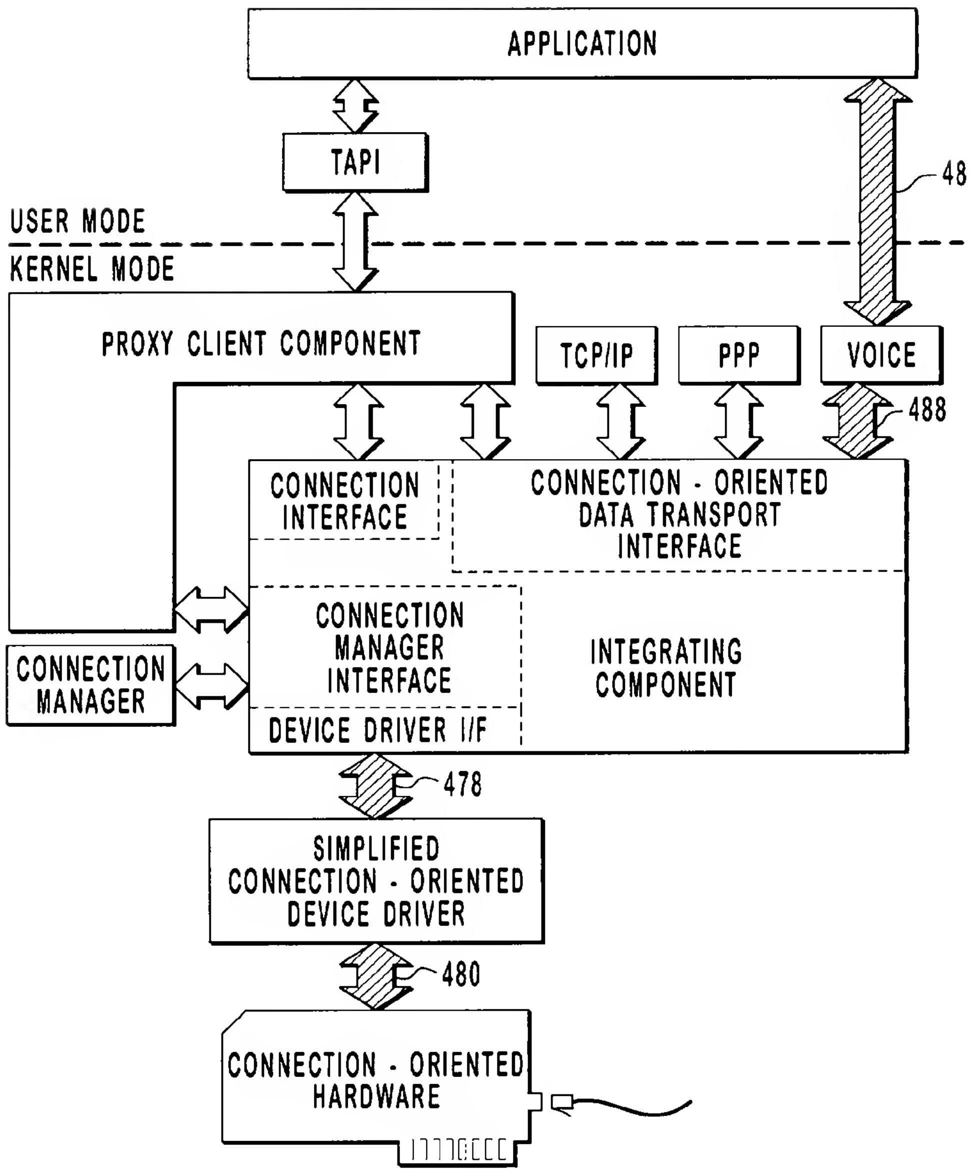

APPLICATION
TAPI
486
USER MODE
KERNEL MODE
PROXY CLIENT COMPONENT
TCP/IP
PPP
VOICE
488
CONNECTION INTERFACE
CONNECTION - ORIENTED DATA TRANSPORT INTERFACE
CONNECTION MANAGER INTERFACE
INTEGRATING COMPONENT
CONNECTION MANAGER
DEVICE DRIVER I/F
478
SIMPLIFIED CONNECTION - ORIENTED DEVICE DRIVER
480
CONNECTION - ORIENTED HARDWARE

FIG. 19C

United States Patent & Trademark Office

Office of Initial Patent Examination -- Scanning Division

Application deficiencies found during scanning:

1 Application papers are not suitable for scanning and are not in compliance with 37 CFR 1.52 because:

- ☐ All sheets must be the same size and either A4 (21 cm x 29.7 cm) or 8-1/2" x 11"
  Pages ______________________ do not meet these requirements.
- ☐ Papers are not flexible, strong, smooth, non-shiny, durable, and white.
- ☐ Papers are not typewritten or mechanically printed in permanent ink on one side.
- ☐ Papers contain improper margins. Each sheet must have a left margin of at least 2.5 cm (1") and top, bottom and right margins of at least 2.0 cm (3/4").
- ☐ Papers contain hand lettering.

2 Drawings are not in compliance and were not scanned because:

- ☐ The drawings or copy of drawings are not suitable for electronic reproduction.
- ☐ All drawings sheets are not the same size. Pages must be either A4 (21 cm x 29.7 cm) or 8-1/2" x 11"
- ☐ Each sheet must include a top and left margin of at least 2.5 cm (1"), a right margin of at least 1.5 cm (9/16") and a bottom margin of at least 1.0 cm (3/8").

3 Page(s) ________________ are not of sufficient clarity, contrast and quality for electronic reproduction.

4 Page(s) ________________ are missing.

5 OTHER No Declaration